Package 'sparrow'

November 7, 2025

Type Package

Title Take command of set enrichment analyses through a unified interface

Version 1.17.0

Description Provides a unified interface to a variety of GSEA techniques from different bioconductor packages. Results are harmonized into a single object and can be interrogated uniformly for quick exploration and interpretation of results. Interactive exploration of GSEA results is enabled through a shiny app provided by a sparrow.shiny sibling package.

URL https://github.com/lianos/sparrow

BugReports https://github.com/lianos/sparrow/issues

Depends R (>= 4.0)

Imports babelgene (>= 21.4), BiocGenerics, BiocParallel, BiocSet, checkmate, circlize, ComplexHeatmap (>= 2.0), data.table (>= 1.10.4), DelayedMatrixStats, edgeR (>= 3.18.1), ggplot2 (>= 2.2.0), graphics, grDevices, GSEABase, irlba, limma, Matrix, methods, plotly (>= 4.9.0), stats, utils, viridis

Suggests AnnotationDbi, BiasedUrn, Biobase (>= 2.24.0), BiocStyle, DESeq2, dplyr, dtplyr, fgsea, GSVA, GO.db, goseq, hexbin, KernSmooth, knitr, magrittr, matrixStats, msigdbr (>= 10.0), orthogene, PANTHER.db (>= 1.0.3), R.utils, reactome.db, rmarkdown, SummarizedExperiment, statmod, stringr, testthat, webshot

biocViews GeneSetEnrichment, Pathways

BiocType Software

VignetteBuilder knitr

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Collate 'AllClasses.R' 'AllGenerics.R' 'GeneSetDb-class.R'
'GeneSetDb-methods.R' 'SparrowResult-methods.R' 'aaa.R'
'bioc-accessors.R' 'calculateIndividualLogFC.R'
'convertIdentifiers.R' 'validateInputs.R' 'do.camera.R'
'do.cameraPR.R' 'do.fgsea.R' 'do.fry.R' 'do.geneSetTest.R'
'do.goseq.R' 'do.logFC.R' 'do.ora.R' 'do.roast.R' 'do.romer.R'
'do.svdGeneSetTest.R' 'geneSetSummaryByGenes.R' 'get-kegg.R'
'get-msigdb.R' 'get-panther.R' 'get-reactome.R'
'gsea-helpers.R' 'package.R' 'plots-corplot.R'
'plots-interactive.R' 'plots-mgheatmap.R' 'plots-mgheatmap2.R'
'renameCollections.R' 'renameRows.R' 'scale_rows.R'
'scoreSingleSamples.R' 'seas.R'
'single-sample-scoring-methods.R' 'species.R'
'testing-helpers.R' 'utilities.R' 'volcano_plot.R' 'zzz.R'

RoxygenNote 7.3.2

Roxygen list(markdown = TRUE)

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

git_branch devel

git_last_commit 03b668b

git_last_commit_date 2025-10-29

Repository Bioconductor 3.23

Date/Publication 2025-11-06

Author Steve Lianoglou [aut, cre] (ORCID:
<https://orcid.org/0000-0002-0924-1754>),
Arkadiusz Gladki [ctb],
Aratus Informatics, LLC [fnd] (2023+),
Denali Therapeutics [fnd] (2018-2022),
Genentech [fnd] (2014 - 2017)

Maintainer Steve Lianoglou <slianoglou@gmail.com>

Contents

addGeneSetMetadata *Add metadata at the geneset level.*

Description

This function adds/updates columns entries in the geneSets(gdb) table. If there already are defined meta values for the columns of meta in x, these will be updated with the values in meta.

Usage

```
addGeneSetMetadata(x, meta, ...)
```

Arguments

x	a GeneSetDb object
meta	a data.frame-like object with "collection", "name", and an arbitrary amount of columns to add as metadata for the genesets.
...	not used yet

Details

TODO: should this be a setReplaceMethod, Issue #13 (?) https://github.com/lianos/multiGSEA/issues/13

Value

the updated GeneSetDb object x.

Examples

```
gdb <- exampleGeneSetDb()
meta.info <- transform(
  geneSets(gdb)[, c("collection", "name")],
  someinfo = sample(c("one", "two"), nrow(gdb), replace = TRUE))
gdb <- addGeneSetMetadata(gdb, meta.info)
```

all.equal.GeneSetDb *Checks equality (feature parity) between GeneSetDb objects*

Description

Checks equality (feature parity) between GeneSetDb objects

Usage

```
## S3 method for class 'GeneSetDb'
all.equal(target, current, features.only = TRUE, ...)
```

Arguments

target	The reference GeneSetDb to compare against
current	The GeneSetDb you wan to compare
features.only	Only compare the "core" columns of target@db and target@table. It is possible that you added additional columns (to keep track of symbols in target@db, for instance) that you want to ignore for the purposes of the equality test.
...	moar args.

Value

TRUE if equal, or character vector of messages if not.

---

annotateGeneSetMembership

Annotates rows of a data.frame with geneset membership from a GeneSetDb

---

Description

This is helpful when you don't have a monsterly sized GeneSetDb. There will be as many new columns added to x as there are active genesets in gdb.

Usage

```
annotateGeneSetMembership(x, gdb, x.ids = NULL, ...)
```

Arguments

x	A data.frame with genes/features in rows
gdb	A GeneSetDb() object with geneset membership
x.ids	The name of the column in x that holds the feautre id's in x that match the feature_id's in gdb, or a vector of id's to use for each row in x that represent these.
...	parameters passed down into incidenceMatrix()

Value

Returns the original x with additional columns: each is a logical vector that indicates membership for genesets defined in gdb.

Examples

```
vm <- exampleExpressionSet()
gdb <- exampleGeneSetDb()
mg <- seas(vm, gdb, design = vm$design, contrast = 'tumor')
lfc <- logFC(mg)
annotated <- annotateGeneSetMembership(lfc, gdb, 'feature_id')

## Show only genes that are part of 'HALLMARK_ANGIOGENESIS' geneset
angio <- subset(annotated, `c2;;BIOCARTA_AGPCR_PATHWAY`)
```

---

calculateIndividualLogFC

Utility function to run limma differential expression analysis

---

Description

Utility function to run limma differential expression analysis

Usage

```
calculateIndividualLogFC(
  x,
  design,
  contrast = ncol(design),
  robust.fit = FALSE,
  robust.eBayes = FALSE,
  trend.eBayes = FALSE,
  treat.lfc = NULL,
  weights = NULL,
  confint = TRUE,
  with.fit = FALSE,
  use.qlf = TRUE,
  ...,
  xmeta. = NULL,
  as.dt = FALSE
)
```

Arguments

x	The expression object. This can be 1 column matrix if you are not running any analysis, and this function essentially is just a "pass through"
design	The design matrix for the experiment
contrast	The contrast you want to test and provide stats for. By default this tests the last column of the design matrix. If you want to test a custom contrast, this can be a contrast vector, which means that it should be as long as ncol(design) and it most-often sum to one. In the future, the user will be able to specify a range of coefficients over design to perform an ANOVA analysis, cf. Issue #11 (https://github.com/lianos/multiGSEA/issues/11).

robust.fit	The value of the robust parameter to pass down to the limma::lmFit() function. Defaults to FALSE.
robust.eBayes	The value of the robust parameter to pass down to the limma::eBayes()] function.
trend.eBayes	The value of the trend parameter to pass down to the limma::eBayes() function.
treat.lfc	If this is numeric, this activates limma's "treat" functionality and tests for differential expression against this specified log fold change threshold. This defaults to NULL.
weights	an option matrix of weights to use in limma::lmFit(). If x is an EList already, and x$weights is already defined, this argument will be ignored.
confint	add confidence intervals to topTable output (default TRUE)? Ignored if x is a DGEList.
with.fit	If TRUE, this function returns a list object with both the fit and the table of logFC statistics, otherwise just the logFC statistics table is returned.
use.qlf	If TRUE (default), will use edgeR's quasilikelihood framework for analysis, otherwise uses glmFit/glmTest.
...	parameters passed down into the relevant limma/edgeR based functions.
xmeta.	a data.frame to add meta data (symbol, primarly) to the outgoing logFC data.frame. This is used when x was a vector (pre-ranked).
as.dt	If FALSE (default), the data.frame like thing that this funciton returns will be set to a data.frame. Set this to TRUE to keep this object as a data.table

Details

This function fits linear modles (or glms) to perform differential expression analyses. If the x object is a DGEList the analysis will be performed using edgeR's quasi-likelihood framework, otherwise limma will be used for all other scenarios.

If x is a edgeR::DGEList() we require that edgeR::estimateDisp() has already been called. If you prefer to analyze rnaseq data using voom, be sure that x is the object that has been returned from a call to limma::voom() (or limma::voomWithQualityWeights().

The documentation here is speaking the language of a "limma" analysis, however for each parameter, there is an analagous function/parameter that will be delegated to.

Lastly, if x is simply a single column matrix, we assume that we are just passing a single pre-ranked vector of statistics through sparrow::seas's analysis pipelines (for use in methods like "fgsea", "cameraPR", etc.), and a logFC-like data.frame is constructed with these statistics in the logFC and t columns.

Value

If with.fit == FALSE (the default) a data.table of logFC statistics for the contrast under test. Otherwise, a list is returned with $result containing the logFC statistics, and $fit has the limma fit for the data/design/contrast under test.

Examples

```
vm <- exampleExpressionSet(do.voom = TRUE)
lfc <- calculateIndividualLogFC(vm, vm$design, "tumor")
```

`collectionMetadata` *Gene Set Collection Metadata*

Description

Associates key:value metadata to a gene set collection of a `GeneSetDb()`.

Usage

```
collectionMetadata(x, collection, name, ...)

geneSetURL(x, i, j, ...)

featureIdType(x, i, ...)

featureIdType(x, i) <- value

## S4 method for signature 'GeneSetDb,missing,missing'
collectionMetadata(x, collection, name, as.dt = FALSE)

## S4 method for signature 'GeneSetDb,character,missing'
collectionMetadata(x, collection, name, as.dt = FALSE)

## S4 method for signature 'GeneSetDb,character,character'
collectionMetadata(x, collection, name, as.dt = FALSE)

## S4 method for signature 'GeneSetDb'
geneSetURL(x, i, j, ...)

## S4 replacement method for signature 'GeneSetDb'
featureIdType(x, i) <- value

## S4 method for signature 'GeneSetDb'
featureIdType(x, i, ...)

addCollectionMetadata(
  x,
  xcoll,
  xname,
  value,
  validate.value.fn = NULL,
  allow.add = TRUE
)
```

```
## S4 method for signature 'SparrowResult'
geneSetURL(x, i, j, ...)
```

Arguments

x	GeneSetDb()
collection	The geneset collection to to query
name	The name of the metadata variable to get the value for
...	not used yet
i, j	The collection,name compound key identifier of the gene set
value	The value of the metadata variable
as.dt	If FALSE (default), the data.frame like thing that this funciton returns will be set to a data.frame. Set this to TRUE to keep this object as a data.table
xcoll	The collection name
xname	The name of the metadata variable
validate.value.fn	If a function is provided, it is run on value and msut return TRUE for addition to be made
allow.add	If FALSE, this xcoll,xname should be in the GeneSetDb already, and this will fail because something is deeply wrong with the world

Details

The design of the GeneSetDb is such that we assume that groups of gene sets are usually defined together and will therefore share similar metadata. These groups of gene sets will fall into the same "collection", and, therefore, metadata for particular gene sets are tracked at the collection level.

Types of metadata being referred to could be things like the organism that a batch of gene sets were defined in, the type of feature identifiers that a collection of gene sets are using (ie. GSEABase::EntrezIdentifier()) or a URL pattern that combines the collection,name compound key that one can browse to in order to find out more information about the gene set.

There are explicit helper functions that set and get these aforementioned metadata, namely featureIdType(), geneSetCollectionURLfunction(), and geneSetURL(). Aribtrary metadata can be stored at the collection level using the addCollectionMetadata() function. More details are provided below.

Value

A character vector of URLs for each of the genesets identified by i, j. NA is returned for genesets i,j that are not found in x.

The updated GeneSetDb.

Methods (by class)

- `collectionMetadata(x = GeneSetDb, collection = missing, name = missing)`: Returns metadata for all collections
- `collectionMetadata(x = GeneSetDb, collection = character, name = missing)`: Returns all metadata for a specific collection
- `collectionMetadata(x = GeneSetDb, collection = character, name = character)`: Returns the `name` metadata value for a given `collection`.
- `geneSetURL(GeneSetDb)`: returns the URL for a geneset
- `featureIdType(GeneSetDb) <- value`: sets the feature id type for a collection
- `featureIdType(GeneSetDb)`: retrieves the feature id type for a collection
- `geneSetURL(SparrowResult)`: returns the URL for a geneset from a SparrowResult object

Gene Set URLs

A URL function can be defined per collection that takes the collection,name compound key and generates a URL for the gene set that the user can browse to for futher information. For instance, the `geneSetCollectionURLfunction()` for the MSigDB collections are defined like so:

```
url.fn <- function(collection, name) {
  url <- 'http://www.broadinstitute.org/gsea/msigdb/cards/%s.html'
  sprintf(url, name)
}
gdb <- getMSigGeneSetDb('H')
geneSetCollectionURLfunction(gdb, 'H') <- url.fn
```

In this way, a call to `geneSetURL(gdb, 'H', 'HALLMARK_ANGIOGENESIS')` will return http://www.broadinstitute.org/gsea/ms

This function is vectorized over `i` and `j`

Feature ID Types

When defining a set of gene sets in a collection, the identifiers used must be of the same type. Most often you'll probably be working with Entrez identifiers, simply because that's what most of the annotations work with.

As such, you'd define that your collection uses geneset identifiers like so:

```
gdb <- getMSigGeneSetDb('H')
featureIdType(gdb, 'H') <- "ensembl"
## or, equivalently (but you don't want to use this)
gdb <- addCollectionMetadata(gdb, 'H', 'id_type', "ensembl")
```

Adding arbitrary collectionMetadata

Adds arbitrary metadata to a gene set collection of a GeneSetDb

Note that this is not a replacement method! You must catch the returned object to keep the one with the updated `collectionMetadata`. Although this function is exported, I imagine this being used mostly through predefined replace methods that use this as a utility function, such as the replacement methods `featureIdType<-`, `geneSetURLfunction<-`, etc.

```
gdb <- getMSigGeneSetDb('H')
gdb <- addCollectionMetadata(gdb, 'H', 'foo', 'bar')
```

Examples

```
gdb <- exampleGeneSetDb()

# Gene Set URLs
geneSetURL(gdb, 'c2', 'BIOCARTA_AGPCR_PATHWAY')
geneSetURL(gdb, c('c2', 'c7'),
           c('BIOCARTA_AGPCR_PATHWAY', 'GSE14308_TH2_VS_TH1_UP'))

# feature id types
featureIdType(gdb, "c2") <- "entrez" # GSEABase::EntrezIdentifier()
featureIdType(gdb, "c2")

## Arbitrary metadata
gdb <- addCollectionMetadata(gdb, 'c2', 'foo', 'bar')
cmh <- collectionMetadata(gdb, 'c2', as.dt = TRUE) ## print this to see
```

---

combine,GeneSetDb,GeneSetDb-method

Combines two GeneSetDb objects together

Description

Combines two GeneSetDb objects together

Usage

```
## S4 method for signature 'GeneSetDb,GeneSetDb'
combine(x, y, ...)
```

Arguments

x	a GeneSetDb object
y	a GeneSetDb object
...	more things

Value

a new GeneSetDb that contains all genesets from x and y

Examples

```
gdb1 <- exampleGeneSetDb()
gdb2 <- GeneSetDb(exampleGeneSetDF())
gdb <- combine(gdb1, gdb2)
```

combine,SparrowResult,SparrowResult-method
Combines two SparrowResult objects together.

Description

This would be useful when you want to add a GSEA result to an already existing one. append would be more appropriate, but ...

Usage

```
## S4 method for signature 'SparrowResult,SparrowResult'
combine(x, y, rename.x = NULL, rename.y = NULL, ...)
```

Arguments

x	A SparrowResult object
y	A SparrowResult object
rename.x	A named vector that used to match resultNames(x) and remane them to something different. names(rename.x) should match whatever you want to change in resultNames(x), and the values are the new names of the result.
rename.y	Same as rename.x, but for the results in y.
...	more things

Details

When would you want to do that? Imagine a shiny app that drives sparrow. You might want to present the results of each analysis as they come "online", so you would run them independently and make them available to the user immediately after they each finish (ie. in combination with the promises package).

Value

A combined SparrowResult object

Examples

```
mg1 <- exampleSparrowResult()
mg2 <- exampleSparrowResult()
mgc <- combine(mg1, mg2)
```

`conform` *(Re)-map geneset IDs to the rows in an expression object.*

Description

`conform`-ing, a `GeneSetDb` to a target expression object is an important step required prior to perform any type of GSEA. This function maps the featureIds used in the GeneSetDb to the elements of a target expression object (ie. the rows of an expression matrix, or the elements of a vector of gene-level statistics).

After `conform`-ation, each geneset in the `GeneSetDb` is flagged as active (or inactive) given the number of its features that are successfully mapped to `target` and the minimum and maximum number of genes per geneset required as specified by the `min.gs.size` and `max.gs.size` parameters, respectively.

Only genesets that are marked with `active = TRUE` will be used in any downstream gene set operations.

Usage

```
conform(x, ...)

unconform(x, ...)

## S4 method for signature 'GeneSetDb'
conform(
  x,
  target,
  unique.by = c("none", "mean", "var"),
  min.gs.size = 2L,
  max.gs.size = Inf,
  match.tolerance = 0.25,
  ...
)

## S4 method for signature 'GeneSetDb'
unconform(x, ...)

is.conformed(x, to)
```

Arguments

`x`	The GeneSetDb
`...`	moar args
`target`	The expression object/matrix to conform to. This could also just be a character vector of IDs.
`unique.by`	If there are multiple rows that map to the identifiers used in the genesets, this is a means to pick the single row for that ID

min.gs.size	Ensure that the genesets that make their way to the GeneSetDb@table are of a minimum size
max.gs.size	Ensure that the genesets that make their way to the GeneSetDb@table are smaller than this size
match.tolerance	Numeric value between [0,1]. If the fraction of feature_ids used in x that match rownames(y) is below this number, a warning will be fired.
to	the object to test conformation to

Value

A GeneSetDb() that has been matched/conformed to an expression object target y.

Functions

- is.conformed(): Checks to see if GeneSetDb x is conformed to a target object to

Related Functions

- unconform(): Resets the conformation mapping.
- is.conformed(): If to is missing, looks for evidence that conform has been called (at all) on x. If to is provided, specifically checks that x has been conformed to the target object to.

Examples

```
es <- exampleExpressionSet()
gdb <- exampleGeneSetDb()
head(geneSets(gdb))
gdb <- conform(gdb, es)
## Note the updated values `active` flag, and n (the number of features
## mapped per gene set)
head(geneSets(gdb))
```

---

conversion *Convert a GeneSetDb to other formats.*

Description

As awesome as a GeneSetDb is, you might find a time when you'll need your gene set information in an other format. To do that, we provide the following functions:

- as(gdb, "BiocSetf"): convert to a BiocSet::BiocSet().
- as(gdb, "GeneSetCollection"): Convert to a GSEABase::GeneSetCollection() object.
- as.data.(table|frame)(gdb): Perhaps the most natural format to convert to in order to save locally and examine outside of Bioconductor's GSEA universe, but not many other tools accept gene set definitions in this format.
- as.list(gdb): A named list of feature identifiers. This is the format that many of the limma gene set testing methods use

Usage

```
## S3 method for class 'GeneSetDb'
as.data.table(
  x,
  keep.rownames = FALSE,
  value = c("feature_id", "x.id", "x.idx"),
  active.only = is.conformed(x),
  ...
)

## S3 method for class 'GeneSetDb'
as.data.frame(
  x,
  row.names = NULL,
  optional = NULL,
  value = c("feature_id", "x.id", "x.idx"),
  active.only = is.conformed(x),
  ...
)
```

Arguments

x	A GeneSetDb object
keep.rownames	included here just for consistency with data.table::as.data.table, but it is not used
value	The value type to export for the feature ids, defaults to "feature_id".
active.only	If the GeneSetDb is conformed, do you want to only return the features and genests that match target and are "active"?
...	pass through arguments (not used)
row.names, optional	included here for consistency with as.data.frame generic function definition, but these are not used.

Details

The as.* functions accept a value parameter which indicates the type of IDs you want to export in the conversion:

- "feature_id": The ID used as originally entered into the GeneSetDb.
- "x.idx": Only valid if the GeneSetDb x has been conform-ed to an expession container. This option will export the features as the integer rows of the expression container.
- "x.id": Only valid if the GeneSetDb x has been conform-ed. The target expression container might use feature identifiers that are different than what is in the GeneSetDb. If an active featureMap is set on the GeneSetDb, this will convert the original feature identifiers into a different target space (entrez to ensembl, for instance). Using this option, the features will be provided in the target space.

Value

a converted GeneSetDb

Functions

- as.data.frame(GeneSetDb): convert a GeneSetDb to data.frame

Examples

```
es <- exampleExpressionSet()
gdb <- conform(exampleGeneSetDb(), es)
bs <- as(gdb, "BiocSet")
gdf <- as.data.frame(gdb)
gdb <- conform(gdb, es)
gdfi <- as.data.frame(gdb, value = 'x.idx')
gdl <- as.list(gdb)
```

---

convertIdentifiers *Converts internal feature identifiers in a GeneSetDb to a set of new ones.*

Description

The various GeneSetDb data providers (MSigDb, KEGG, etc). limit the identifier types that they return. Use this function to map the given identifiers to whichever type you like.

Usage

```
convertIdentifiers(
  x,
  from = NULL,
  to = NULL,
  id.type = c("ensembl", "entrez", "symbol"),
  xref = NULL,
  extra.cols = NULL,
  allow.cartesian = FALSE,
  method = c("orthogene", "babelgene"),
  min_support = 3,
  top = TRUE,
  ...
)

## S4 method for signature 'BiocSet'
convertIdentifiers(
  x,
  from = NULL,
  to = NULL,
```

```
  id.type = c("ensembl", "entrez", "symbol"),
  xref = NULL,
  extra.cols = NULL,
  allow.cartesian = FALSE,
  method = c("orthogene", "babelgene"),
  min_support = 3,
  top = TRUE,
  ...
)

## S4 method for signature 'GeneSetDb'
convertIdentifiers(
  x,
  from = NULL,
  to = NULL,
  id.type = c("ensembl", "entrez", "symbol"),
  xref = NULL,
  extra.cols = NULL,
  allow.cartesian = FALSE,
  method = c("orthogene", "babelgene"),
  min_support = 3,
  top = TRUE,
  ...
)
```

Arguments

x	The GeneSetDb with identifiers to convert
from, to	If you are doing identifier and/orspecies conversion using babelgene, to is the species you want to convert to, and from is the species of x. If you are only doing id type conversion within the same species, specify the current species in from. If you are providing a data.frame map of identifiers in xref, to is the name of the column that holds the new identifiers, and from is the name of the column that holds the current identifiers.
id.type	If you are using babelgene conversion, this specifies the type of identifier you want to convert to. It can be any of "ensembl", "entrez", or "symbol".
xref	a data.frame used to map current identifiers to target ones.
extra.cols	a character vector of columns from to to add to the features of the new GeneSetDb. If you want to keep the original identifiers of the remapped features, include "original_id" as one of the values here.
allow.cartesian	a boolean used to temporarily set the datatable.allow.cartesian global option. If you are doing a 1:many map of your identifiers, you may trigger this error. You can temporarily turn this option/error off by setting allow.cartesian = TRUE. The option will be restored to its "pre-function call" value on.exit.
method	The method used to convert identifers, either "orthogene" or "babelgene". "orthogene" (the default) is more powerful, supports more organisms, and

(unlike "babelgene") can map between any two arbitrary species – babelgene requires one of the species in the mapping to be human. The downside to "orthogene" is that you need internet access to run.

min_support, top
: Parameters used in the internal call to babelgene::orthologs()

...
: pass through args (not used)

Details

For best results, provide your own identifier mapping reference, but we provide a convenience wrapper around the babelgene::orthologs() function to change between identifier types and species.

When there are multiple target id's for the source id, they will all be returned. When there is no target id for the source id, the soucre feature will be axed.

Value

A new GeneSetDb object with converted identifiers. We try to retain any metadata in the original object, but no guarantees are given. If id_type was stored previously in the collectionMetadata, that will be dropped.

Methods (by class)

- convertIdentifiers(BiocSet): converts identifiers in a BiocSet
- convertIdentifiers(GeneSetDb): converts identifiers in a GeneSetDb

Custom Mapping

You need to provide a data.frame via the xref paramater that has a column for the current identifiers and another column for the target identifiers. The columns are specified by the from and to paramters, respectively.

Convenience identifier and species mapping

If you don't provide a data.frame, you can provide a species name. We will rely on the {babelgene} package for the conversion, so you will have to provide a species name that it recognizes.

Species and Identifier Conversion via babelgene

We plan to provide a quick wrapper to babelgene's ortholog mapping function to make identifier conversion a easier through this function. You can track this in sparrow issue #2.

Species and Identifier Conversion via orthogene

Babelgene is great, but does not support all species (like cynos), but we can rely on the orthogene package for that. The downside to orthogene is that it requires online acces.

Examples

```
# You can convert the identifiers within a GeneSetDb to some other type
# by providing a "translation" table. Check out the unit tests for more
# examples.
gdb <- exampleGeneSetDb() # this has no symbols in it

# Define a silly conversion table.
xref <- data.frame(
  current_id = featureIds(gdb),
  new_id = paste0(featureIds(gdb), "_symbol"))
gdb2 <- convertIdentifiers(gdb, from = "current_id", to = "new_id",
                           xref = xref, extra.cols = "original_id")
geneSet(gdb2, name = "BIOCARTA_AGPCR_PATHWAY")

# Convert entrez to ensembl id's using babelgene
## Not run:
# The conversion functionality via babelgene isn't yet implemented, but
# will look like this.

# 1. convert the human entrez identifiers to ensembl
gdb.ens <- convertIdentifiers(gdb, "human", id.type = "ensembl")

# 2. convert the human entrez to mouse entrez
gdb.entm <- convertIdentifiers(gdb, "human", "mouse", id.type = "entrez")

# 3. convert the human entrez to mouse ensembl
gdb.ensm <- convertIdentifiers(gdb, "human", "mouse", id.type = "ensembl")

## End(Not run)
```

---

corplot *Plots the correlation among the columns of a numeric matrix.*

Description

We assume that this is a sample x gene expression matrix, but it can (of course) be any numeric matrix of your choosing. The column names appear in the main diagonal of the plot. Note that you might prefer the corrplot package for similar functionality, and this functionality is intentionally named different from that..

Usage

```
corplot(
  E,
  title,
  cluster = FALSE,
  col.point = "#00000066",
  diag.distro = TRUE,
  smooth.scatter = nrow(E) > 400,
```

```
  max.cex.cor = NULL,
  ...
)
```

Arguments

E	the matrix used to plot a pairs correlation plot. The vectors used to assess all pairwise correlation should be *in the columns* of the matrix.
title	The title of the plot
cluster	logical indicating whether or not to shuffle genes around into some clustering.
col.point	the color of the points in the scatterplots
diag.distro	show the distribution of values on the diagnols?
smooth.scatter	boolean to indicate wether to use a normal scatter, or a graphics::smoothScatter(). Defaults to TRUE if nrow(E) > 400
max.cex.cor	the numeric value defining the maximum text size (cor) in the correlation panel. By default there is no limit on the maximum text size and the text size is calculated with 0.8 / strwidth(text). With max.cex.cor defined the text size is calculated as min(0.8 / strwidth(text), max.cex.cor).
...	pass through arguments to internal panel functions

Details

TODO: Add with.signature parameter to allow a box to plot the signature score of all genes in E.

Value

nothing, just creates the plot

See Also

The corrplot package

Examples

```
x <- matrix(rnorm(1000), ncol=5)
corplot(x)
```

---

eigenWeightedMean *Single sample gene set score by a weighted average of the genes in geneset*

Description

Weights for the genes in x are calculated by the percent of which they contribute to the principal component indicated by eigengene.

Usage

```
eigenWeightedMean(
  x,
  eigengene = 1L,
  center = TRUE,
  scale = TRUE,
  uncenter = center,
  unscale = scale,
  retx = FALSE,
  weights = NULL,
  normalize = FALSE,
  all.x = NULL,
  ...,
  .drop.sd = 1e-04
)
```

Arguments

x	An expression matrix of genes x samples. When using this to score geneset activity, you want to reduce the rows of x to be only the genes from the given gene set.
eigengene	the PC used to extract the gene weights from
center, scale	center and/or scale data before scoring?
uncenter, unscale	uncenter and unscale the data data on the way out? Defaults to the respective values of center and scale
retx	Works the same as retx from prcomp. If TRUE, will return a retpcax matrix that has the rotated variables.
weights	a user can pass in a prespecified set of waits using a named numeric vector. The names must be a superset of rownames(x). If this is NULL, we calculate the "eigenweights".
normalize	If TRUE, each score is normalized to a randomly selected geneset score. The size of the randomly selected geneset is the same as the corresponding geneset. This only works with the "ewm" method when unscale and uncenter are TRUE. By default, this is set to FALSE, and normalization does not happen. Instead of passing in TRUE, the user can pass in a vector of gene names (identifiers) to be considered for random geneset creation. If no genes are provided, then all genes in y are fair game.
all.x	if the user is trying to normalize these scores, an expression matrix that has superset of the control genes needs to be provided, where the columns of all.x must correspond to this in x.
...	these aren't used in here
.drop.sd	When zero-sd (non varying) features are scaled, their values are NaN. When the Features with rowSds < this threshold (default 1e-4) are identified, and their scaled values are set to 0.

Details

You will generally want the rows of the gene x sample matrix "xto be z-transformed. If it is not already, ensure th and 'scale' are set to 'TRUE'.

When uncenter and/or unscale are FALSE, it means that the scores should be applied on the centered or scaled values, respectively.

Value

A list of useful transformation information. The caller is likely most interested in the $score vector, but other bits related to the SVD/PCA decomposition are included for the ride.

Normalization

Scores can be normalized against a set of control genes. This results in negative and postiive sample scores. Positive scores are ones where the specific geneset score is higher than the aggregate control-geneset score.

Genes used for the control set can either be randomly sampled from the rows of the all.x expression matrix (when normalize = TRUE), or explicitly specified by a row-identifier character vectore passed to the normalize parameter. In both cases, the code prefers to select a random-control geneset to be of equal size as nrow(x). If that's not possible, we use as many genes as we can get.

Note that normalization requires an expression matrix to be passed into the all.x parameter, whose columns match 1:1 to the columns in x. Calling scoreSingleSamples() with method = "ewm", normalize = TRUE handles this transparently.

This idea to implement this method of normalizatition was inspired from the ctrl.score normalization found in Seurat's AddModuleScore() function.

See Also

scoreSingleSamples

Examples

```
vm <- exampleExpressionSet(do.voom=TRUE)
gdb <- conform(exampleGeneSetDb(), vm)
features <- featureIds(gdb, 'c2', 'BURTON_ADIPOGENESIS_PEAK_AT_2HR',
                       value='x.idx')
ewm <- eigenWeightedMean(vm[features,])
scores <- ewm$score

## Use scoreSingleSamples to facilitate scoring of all gene sets
scores.all <- scoreSingleSamples(gdb, vm, 'ewm')
s2 <- with(subset(scores.all, name == 'BURTON_ADIPOGENESIS_PEAK_AT_2HR'),
           setNames(score, sample_id))
all.equal(s2, scores)
```

encode_gskey *Converts collection,name combination to key for geneset*

Description

The "key" form often comes out as rownames to matrices and such, or particularly for sending genesets down into wrapped methods, like do.camera.

splt_gskey is the inverse function of encode_gskey()

Usage

```
encode_gskey(x, y, sep = ";;")

split_gskey(x, sep = ";;")
```

Arguments

x	a character vector of encoded geneset keys from encode_gskey()
y	if x is a data.frame: nothing, otherwise a character vector of geneset names
sep	the separator used in the encoding of geneset names

Value

a character vector

a data.frame with (collection,name) columns

Examples

```
gdf <- exampleGeneSetDF()
gskeys <- encode_gskey(gdf)
gscols <- split_gskey(gskeys)
```

exampleExpressionSet *Functions that load data for use in examples and testing.*

Description

We provide examplar expression data (counts or voomed) as well as exemplar gene sets in different forms.

Usage

```
exampleExpressionSet(
  dataset = c("tumor-vs-normal", "tumor-subtype"),
  do.voom = TRUE
)

exampleGeneSets(x, unlist = !missing(x))

exampleGeneSetDb()

exampleBiocSet()

exampleGeneSetDF()

exampleSparrowResult(cached = TRUE, methods = c("cameraPR", "fry"))

exampleDgeResult(
  species = "human",
  id.type = c("entrez", "ensembl"),
  induce.bias = NULL
)
```

Arguments

dataset	Character vector indicating what samples wanted, either "tumor-vs-normal" for a tumor vs normal dataset from TCGA, or just the tumor samples from the same annotated with subtype.
do.voom	If TRUE, a voomed EList is returned, otherwise an ExpressionSet of counts.
x	If provided, an expression/matrix object so that the genesets are returned as (integer) index vectors into the rows of x whose rownames match the ids in the geneset.
unlist	return the genesets as nested list of lists (default: TRUE). The top level lists corresponds to the collection, and the lists within each are the inidividual gene sets. If FALSE, a single list of genesets is returned.
cached	If TRUE (default), returns a pre-saved SparrowResult object. Otherwise calculates a fresh one using the methods provided
methods	the methods to use to create a new SparrowResult for.
species	the species to return the example result from (right now, only "human")
id.type	the type of identifiers to use: "entrez" (default) or "ensembl".
induce.bias	We can simulate a bias on the pvalue by the gene's "effective_length" or "AveExpr". These are columns that are included in the output. If NULL, no bias is introduced into the result.

Value

A list of lists of entrezIDs when as == 'lol', or a list of integers into the rows of x.

exampleExpressionSet

The expression data is a subset of the TCGA BRCA indication. Calling `exampleExpressionSet(do.voom = TRUE)` will return a voomed `EList` version of the data. When `do.voom = FALSE`, you will get a DGEList of the counts

exampleGeneSets

Returns gene sets as either a list of feature identifiers. Entrez identifiers are used. If `x` is provided, integers that index into the expression container `x` are used (this is a legacy feature that we should nuke).

exampleGeneSetDb

Returns gene sets as a `GeneSetDb` object

exampleBiocSet

Returns gene sets as a `BiocSet` object

exampleGeneSetDF

Returns a data.frame of gene set definitions. A data.frame of this form can be passed into the `GeneSetDb()` contructor.

Examples

```
vm <- exampleExpressionSet()
head(exampleGeneSets())
```

---

failWith *Utility function to try and fail with grace.*

Description

Inspired from one of Hadley's functions (in plyr or something?)

Usage

```
failWith(
  default = NULL,
  expr,
  frame = parent.frame(),
  message = geterrmessage(),
  silent = FALSE,
  file = stderr()
)
```

Arguments

default	the value to return if expr fails
expr	the expression to take a shot at
frame	the frame to evaluate the expression in
message	the error message to display if expr fails. Deafults to base::geterrmessage()
silent	if TRUE, sends the error message to msg()
file	where msg sends the message

Value

the result of expr if successful, otherwise default value.

Examples

```
# look, this doesn't throw an error, it just returns NULL
x <- failWith(NULL, stop("no error, just NULL"), silent = TRUE)
```

---

featureIdMap *Fetch the featureIdMap for a* GeneSetDb

Description

The GeneSetDb has an internal data structure that is used to cross reference the feature_id's used in the database construction to the features in the expression object that is used to run GSEA methods against.

Usage

```
featureIdMap(x, ...)

## S4 method for signature 'GeneSetDb'
featureIdMap(x, as.dt = FALSE)
```

Arguments

x	the object to retrieve the featureIdMap from
...	pass through arguments
as.dt	If FALSE (default), the data.frame like thing that this funciton returns will be set to a data.frame. Set this to TRUE to keep this object as a data.table

Value

a data.frame of input feature_id's to conformed id's/rows/etc

Methods (by class)

- `featureIdMap(GeneSetDb)`: extract featureIdMap from a GeneSetDb

Examples

```
gdb <- exampleGeneSetDb()
vm <- exampleExpressionSet()
gdb <- conform(gdb, vm)
fmap <- featureIdMap(gdb)
```

`featureIds` *Returns the relevant featureIds for a given geneset.*

Description

Gene sets are defined by the unique compound key consisting of their `collection` and `name`. To fetch the featureIds associated with a specific geneset, you must provide values for `i` and `j`. If these are missing, then a character vector of all the unique feature ids within `x` are returned.

If the GeneSetDb `x` has been conformed to an expression object this will default to return only the feature_id's that are matched to the target expression object, and they will be returned using the same identifiers that the target expression object uses. To change this behavior, tweak the values for the `active.only` and `value` parameters, respectively.

`x` can be either a `GeneSetDb` or a `SparrowResult`. If its the latter, then this call simply delegates to the internal `GeneSetDb`.

Usage

```
featureIds(
  x,
  i,
  j,
  value = c("feature_id", "x.id", "x.idx"),
  active.only = is.conformed(x),
  ...
)

## S4 method for signature 'GeneSetDb'
featureIds(
  x,
  i,
  j,
  value = c("feature_id", "x.id", "x.idx"),
  active.only = is.conformed(x),
  ...
)
```

```
## S4 method for signature 'SparrowResult'
featureIds(
  x,
  i,
  j,
  value = c("feature_id", "x.id", "x.idx"),
  active.only = TRUE,
  ...
)
```

Arguments

x	Object to retrieve the gene set from, either a GeneSetDb or a SparrowResult.
i, j	The collection,name compound key identifier of the gene set
value	What form do you want the id's in? • "feature_id": the IDs used in the original geneset definitions • "x.id": the ids of the features as they are used in the expression object. • "x.idx": The integer index into the expresion object x that the 'GeneSetDb" has been conformed to.
active.only	only look for gene sets that are "active"? Defaults to TRUE if x is conformed to a target expression object, else FALSE. conform() for further details.
...	pass through arguments

Value

A vector of identifiers (or indexes into an expression object, depending on the value argument) for the features in the specified geneset. NA is returned if the geneset is not "active" (ie. listed in geneSets())

Examples

```
gdb <- exampleGeneSetDb()
fids.gs <- featureIds(gdb, 'c2', 'BIOCARTA_AGPCR_PATHWAY')
fids.c2 <- featureIds(gdb, 'c2')
fids.all <- featureIds(gdb)

vm <- exampleExpressionSet(do.voom=TRUE)
gdb <- conform(gdb, vm)
## fewer than before
fids.gs2 <- featureIds(gdb, 'c2', 'BIOCARTA_AGPCR_PATHWAY')
## same as before
fids.gs3 <- featureIds(gdb, 'c2', 'BIOCARTA_AGPCR_PATHWAY', active.only=FALSE)
## returned as row indices into vm
fids.idxs <- featureIds(gdb, 'c2', value='x.idx')
```

geneSet *Fetches information for a gene set*

Description

Gene sets inside a GeneSetDb() are indexed by their collection,name compound key. There is no special class to represent an individual gene set. Instead, gene sets are returned as a data.frame, the rows of which enumerate the features that belong to them.

When x is a SparrowResult(), this function will append the differential expression statistics for the individual features generated across the contrast that defined x.

Usage

```
geneSet(x, i, j, ...)

## S4 method for signature 'GeneSetDb'
geneSet(
  x,
  i,
  j,
  active.only = is.conformed(x),
  with.feature.map = FALSE,
  ...,
  collection = NULL,
  name = NULL,
  as.dt = FALSE
)

## S4 method for signature 'SparrowResult'
geneSet(
  x,
  i,
  j,
  active.only = TRUE,
  with.feature.map = FALSE,
  ...,
  collection = NULL,
  name = NULL,
  as.dt = FALSE
)
```

Arguments

x	Object to retrieve the gene set from, either a GeneSetDb or a SparrowResult.
i, j	The collection,name compound key identifier of the gene set
...	passed down to inner functinos

active.only — only look for gene sets that are "active"? Defaults to TRUE if x is conformed to a target expression object, else FALSE. conform() for further details.

with.feature.map — If TRUE, then details of the feature mapping from the original feature_id space to the target feature space are included (default: FALSE).

collection — using i as the parameter for "collection" isn't intuitive so if speficially set this paramter, it will replace the value for i.

name — the same for the collection:i parameter relationship, but for j:name.

as.dt — If FALSE (default), the data.frame like thing that this funciton returns will be set to a data.frame. Set this to TRUE to keep this object as a data.table

Value

a data.(frame|table) of gene set information. If x is a SparrowResult object, then differential expression statistics are added as columns to this result.

Examples

```
gdb <- exampleGeneSetDb()
geneSet(gdb, "c2", "KOMMAGANI_TP63_GAMMA_TARGETS")
geneSet(gdb, collection = "c2", name = "KOMMAGANI_TP63_GAMMA_TARGETS")
geneSet(gdb, name = "KOMMAGANI_TP63_GAMMA_TARGETS")
```

---

geneSetCollectionURLfunction

Get/set the gene set collection url function for a geneset collection

---

Description

Reference collectionMetadata() for more info.

Usage

```
geneSetCollectionURLfunction(x, i, ...)

geneSetCollectionURLfunction(x, i) <- value

## S4 method for signature 'GeneSetDb'
geneSetCollectionURLfunction(x, i, ...)

## S4 replacement method for signature 'GeneSetDb'
geneSetCollectionURLfunction(x, i) <- value

## S4 method for signature 'SparrowResult'
geneSetCollectionURLfunction(x, i, ...)
```

Arguments

x	The GeneSetDb
i	The collection to get the url function from
...	pass through arguments (not used)
value	the function to set as the geneset url function for the given collection i. This can be an actual function object, or the (string) name of the function to pull out of "the ether" ("pkgname::functionname" can work, too). The latter is preferred as it results in smaller serialized GeneSetDb objects.

Value

the function that maps collection,name combinations to an informational URL.

Methods (by class)

- geneSetCollectionURLfunction(GeneSetDb): returns the gene set collection url function from a GeneSetDb
- geneSetCollectionURLfunction(GeneSetDb) <- value: sets the gene set collection url function for a GeneSetDb : Collection combination.
- geneSetCollectionURLfunction(SparrowResult): return the url function from a SparrowResult object.

Examples

```
gdb <- exampleGeneSetDb()
geneSetCollectionURLfunction(gdb, "c2", "BIOCARTA_AGPCR_PATHWAY")
```

---

geneSetDb *Fetches the GeneSetDb from SparrowResult*

Description

Fetches the GeneSetDb from SparrowResult

Usage

```
geneSetDb(x)
```

Arguments

x	SparrowResult

Value

The GeneSetDb

Examples

```
vm <- exampleExpressionSet(do.voom=TRUE)
gdb <- exampleGeneSetDb()
mg <- seas(vm, gdb, design = vm$design, contrast = 'tumor')
geneSetDb(mg)
```

`GeneSetDb-class` *A container for geneset definitions.*

Description

Please refer to the sparrow vignette (`vignette("sparrow")`), (and the "The GeneSetDb Class" section, in particular) for a more deatiled description of the sematnics of this central data object.

The GeneSetDb class serves the same purpose as the `GSEABase::GeneSetCollection()` class does: it acts as a centralized object to hold collections of Gene Sets. The reason for its existence is because there are things that I wanted to know about my gene set collections that weren't easily inferred from what is essentially a "list of GeneSets" that is the `GeneSetCollection` class.

Gene Sets are internally represented by a `data.table` in "a tidy" format, where we minimally require non `NA` values for the following three `character` columns:

- collection
- name
- feature_id

The (`collection`, `name`) compound key is the primary key of a gene set. There will be as many entries with the same (`collection`, `name`) as there are genes/features in that set.

The `GeneSetDb` tracks metadata about genesets at **the collection** level. This means that we assume that all of the `feature_id`'s used within a collection use the same type of feature identifier (such as a `GSEABase::EntrezIdentifier()`, were defined in the same organism, etc.

Please refer to the "GeneSetDb" section of the vignette for more details regarding the construction and querying of a `GeneSetDb` object.

Usage

```
GeneSetDb(x, featureIdMap = NULL, collectionName = NULL, ...)
```

Arguments

`x`	A `GeneSetCollection`, a "two deep" list of either `GeneSetCollections` or lists of character vectors, which are the gene identifers. The "two deep" list represents the different collections (top level) at the top level, and each such list is a named list itself, which represents the gene sets in the given collection.
`featureIdMap`	A data.frame with 2 character columns. The first column is the ids of the genes (features) used to identify the genes in `gene.sets`, the second second column are IDs that this should be mapped to. Useful for testing probelevel microarray data to gene level gene set information.

`collectionName` If `x` represents a singular collection, ie. a single `GeneSetCollection` or a "one deep" (named (by geneset)) list of genesets, then this parameter provides the name for the collection. If `x` is multiple collections, this can be character vector of same length with the names. In all cases, if a collection name can't be defined from this, then collections will be named anonymously. If a value is passed here, it will overide any names stored in the list of `x`.

`...` these aren't used for anything in particular, but are here to catch extra arguments that may get passed down if this function is part of some call chain.

Details

The functionality in the class is useful for the functionality in this package, but for your own personal usage, you probably want a `{BiocSet}`.

Value

A GeneSetDb object

Slots

`table` The "gene set table": a data.table with geneset information, one row per gene set. Columns include collection, name, N, and n. The end user can add more columns to this data.table as desired. The actual feature IDs are computed on the fly by doing a `db[J(group, id)]`

`db` A `data.table` to hold all of the original geneset id information that was used to construct this `GeneSetDb`.

`featureIdMap` Maps the ids used in the geneset lists to the ids (rows) over the expression data the GSEA is run on

`collectionMetadata` A `data.table` to keep metadata about each individual geneset collection, ie. the user might want to keep track of where the geneset definitions come from. Perhaps a function that parses the collection,name combination to generate an URL that points the user to more information about that geneset, etc.

GeneSetDb Construction

The `GeneSetDb()` constructor is sufficiently flexible enough to create a `GeneSetDb` object from a variety of formats that are commonly used in the bioconductor echosystem, such as:

- `GSEABase::GeneSetCollection()`: If you already have a `GeneSetCollection` on your hands, you can simply pass it to the `GeneSetDb()` constructor.
- list of ids: This format is commonly used to define gene sets in the edgeR/limma universe for testing with camera, roast, romer, etc. The names of the list items are the gene set names, and their values are a character vector of gene identifiers. When it's a single list of lists, you must provide a value for `collectionName`. You can embed multiple collections of gene sets by having a three-deep list-of-lists-of-ids. The top level list define the different collections, the second level are the genesets, and the third level are the feature identifiers for each gene set. See the examples for clarification.

- a data.frame-like object: To keep track of your own custom gene sets, you have probably realized the importance of maintaing your own sanity, and likely have gene sets organized in a table like object that has something like the collection, name, and feature_id required for a GeneSetDb. Simply rename the appropriate columns to the ones prescribed here, and pass that into the constructor. Any other additional columns (symbol, direction, etc.) will be copied into the GeneSetDb.

Interrogating a GeneSetDb

You might wonder what gene sets are defined in a GeneSetDb: see the geneSets() function.

Curious about what features are defined in your GeneSetDb? See the featureIds() function.

Want the details of a particular gene set? Try the geneSet() function. This will return a data.frame of the gene set definition. Calling geneSet() on a SparrowResult() will return the same data.frame along with the differential expression statistics for the individual members of the geneSet across the contrast that was tested in the seas() call that created the SparrowResult().

GeneSetDb manipulation

You can subset a GeneSetDb to include a subset of genesets defined in it. To do this, you need to provide an indexing vector that is as long as length(gdb), ie. the number of gene sets defined in GeneSetDb. You can construct such a vector by performing your boolean logic over the geneSets(gdb) table.

Look at the Examples section to see how this works, where we take the MSIgDB c7 collection (aka. "ImmuneSigDB") and only keep gene sets that were defined in experiments from mouse.

See Also

?conversion

Examples

```
## exampleGeneSetDF provides gene set definitions in "long form". We show
## how this can easily turned into a GeneSetDb from this form, or convert
## it to other forms (list of features, or list of list of features) to
## do the same.
gs.df <- exampleGeneSetDF()
gdb.df <- GeneSetDb(gs.df)

## list of ids
gs.df$key <- encode_gskey(gs.df)
gs.list <- split(gs.df$feature_id, gs.df$key)
gdb.list <- GeneSetDb(gs.list, collectionName='custom-sigs')

## A list of lists, where the top level list splits the collections.
## The name of the collection in the GeneSetDb is taken from this top level
## hierarchy
gs.lol <- as.list(gdb.df, nested=TRUE) ## examine this list-of lists
gdb.lol <- GeneSetDb(gs.lol) ## note that collection is set propperly

## GeneSetDb Interrogation
```

```
gsets <- geneSets(gdb.df)
nkcells <- geneSet(gdb.df, 'cellularity', 'NK cells')
fids <- featureIds(gdb.df)

# GeneSetDb Manipulation ..................................................
# Subset down to only t cell related gene sets
gdb.t <- gdb.df[grepl("T cell", geneSets(gdb.df)$name)]
gdb.t
```

`geneSets` *Fetch the active (or all) gene sets from a GeneSetDb or SparrowResult*

Description

Fetch the active (or all) gene sets from a GeneSetDb or SparrowResult

Usage

```
geneSets(x, ...)

## S4 method for signature 'GeneSetDb'
length(x)

## S4 method for signature 'GeneSetDb'
geneSets(x, active.only = is.conformed(x), ..., as.dt = FALSE)

## S4 method for signature 'GeneSetDb'
nrow(x)

## S4 method for signature 'SparrowResult'
geneSets(x, ..., as.dt = FALSE)
```

Arguments

`x`	Object to retrieve the gene set from, either a `GeneSetDb` or a `SparrowResult`.
`...`	pass through arguments
`active.only`	only look for gene sets that are "active"? Defaults to `TRUE` if `x` is conformed to a target expression object, else `FALSE`. `conform()` for further details.
`as.dt`	If `FALSE` (default), the data.frame like thing that this funciton returns will be set to a data.frame. Set this to `TRUE` to keep this object as a `data.table`

Value

a data.table with geneset information.

Methods (by class)

- `length(GeneSetDb)`: Returns the number of genesets in a GeneSetDb
- `geneSets(GeneSetDb)`: return all genesets from a GeneSetDb
- `nrow(GeneSetDb)`: return number of genesets in GeneSetDb
- `geneSets(SparrowResult)`: return the active genesets from a SparrowResult

Examples

```
gdb <- exampleGeneSetDb()
gs <- geneSets(gdb)
```

---

geneSetsStats *Summarizes useful statistics per gene set from a SparrowResult*

Description

This function calculates the number of genes that move up/down for the given contrasts, as well as mean and trimmed mean of the logFC and t-statistics. Note that the statistics calculated and returned here are purely a function of the statistics generated at the gene-level stage of the analysis.

Usage

```
geneSetsStats(
  x,
  feature.min.logFC = 1,
  feature.max.padj = 0.1,
  trim = 0.1,
  reannotate.significance = FALSE,
  as.dt = FALSE
)
```

Arguments

`x`	A `SparrowResult` object
`feature.min.logFC`	used with `feature.max.padj` to identify the individual features that are to be considered differentially expressed.
`feature.max.padj`	used with `feature.min.logFC` to identify the individual features that are to be considered differentially expressed.
`trim`	The amount to trim when calculated trimmed `t` and `logFC` statistics for each geneset.
`reannotate.significance`	this is internally by the package, and should left as `FALSE` when used by the user.
`as.dt`	If `FALSE` (default), the data.frame like thing that this funciton returns will be set to a data.frame. Set this to `TRUE` to keep this object as a `data.table`

Value

A data.table with statistics at the gene set level across the prescribed contrast run on `x`. These statistics are independent of any particular GSEA method, but rather summarize aggregate shifts of the gene sets individual features. The columns included in the output are summarized below:

- `n.sig`: The number of individual features whose `abs(logFC)` and padj thersholds satisfy the criteria of the `feature.min.logFC` and `feature.max.padj` parameters of the original `seas()` call
- `n.neutral`: The number of individual features whose abs(logFC) and padj thersholds do not satisfy the `feature.*` criteria named above.
- `n.up, n.down`: The number of individual features with `logFC > 0` or `logFC < 0`, respectively, irrespective of the `feature.*` thresholds referenced above.
- `n.sig.up, n.sig.down`: The number of individual features that pass the `feature.*` thresholds and have logFC > 0 or logFC < 0, respectively.
- `mean.logFC, mean.logFC.trim`: The mean (or trimmed mean) of the individual logFC estimates for the features in the gene set. The amount of trim is specified in the `trim` parameter of the `seas()` call.
- `mean.t, mean.t.trim`: The mean (or trimmed mean) of the individual t-statistics for the features in the gene sets. These are `NA` if the input expression object was a `DGEList`.

Examples

```
vm <- exampleExpressionSet(do.voom=TRUE)
gdb <- exampleGeneSetDb()
mg <- seas(vm, gdb, design = vm$design, contrast = 'tumor')
head(geneSetsStats(mg))
```

`geneSetSummaryByGenes` *Summarize geneset:feature relationships for specified set of features*

Description

This function creates a geneset by feature table with geneset membership information for the `features` specified by the user. Only the gene sets that have any of the `features` are included in the table returned.

Usage

```
geneSetSummaryByGenes(
  x,
  features,
  with.features = TRUE,
  feature.rename = NULL,
  ...,
  as.dt = FALSE
)
```

```
## S4 method for signature 'GeneSetDb'
geneSetSummaryByGenes(
  x,
  features,
  with.features = TRUE,
  feature.rename = NULL,
  ...,
  as.dt = FALSE
)

## S4 method for signature 'SparrowResult'
geneSetSummaryByGenes(
  x,
  features,
  with.features = TRUE,
  feature.rename = NULL,
  method = NULL,
  max.p = 0.3,
  p.col = c("padj", "padj.by.collection", "pval"),
  ...,
  as.dt = FALSE
)
```

Arguments

x	GeneSetDb or SparrowResult
features	a character vector of featureIds
with.features	Include columns for features? If x is is a GeneSetDb, these columns are TRUE/FALSE. If x is a SparrowResult object, the values are the logFC of the feature if present in the gene set, otherwise its NA.
feature.rename	if NULL, the feature columns are prefixed with featureId_, if FALSE, no renaming is done. If x is a SparrowResult, then this can be the column name found in logFC(x), in which case the value for the feature from the given column name would be used (setting this to "symbol") would be a common thing to do, for instance.
...	pass through arguments
as.dt	If FALSE (default), the data.frame like thing that this funciton returns will be set to a data.frame. Set this to TRUE to keep this object as a data.table
method	The GSEA method to pull statistics from
max.p	the maximum p-value from the analysis method to allow for the geneSets included in the returned table
p.col	which p-value column to select from: 'padj', 'padj.by.collection', or 'pval'

Value

a data.frame of geneset <-> feature incidence/feature matrix.

Methods (by class)

- `geneSetSummaryByGenes(SparrowResult)`: get geneset:feature incidence table from a SparrowResult, optionally filtered by statistical significance from a given gsea `method`

Examples

```
vm <- exampleExpressionSet(do.voom=TRUE)
gdb <- conform(exampleGeneSetDb(), vm)
mg <- seas(vm, gdb, design = vm$design, contrast = 'tumor')
features <- c("55839", "8522", "29087")
gsm.hit <- geneSetSummaryByGenes(gdb, features)
gsm.fid <- geneSetSummaryByGenes(mg, features, feature.rename=NULL)
gsm.sym <- geneSetSummaryByGenes(mg, features, feature.rename='symbol')
```

`getKeggCollection` *Retrieves the KEGG gene set collection via its REST API*

Description

Uses `limma::getGeneKEGGLinks()` and `limma::getKEGGPathwayNames()` internally.

Usage

```
getKeggCollection(species = "human", id.type = c("ensembl", "entrez"), ...)

getKeggGeneSetDb(species = "human", id.type = c("ensembl", "entrez"), ...)
```

Arguments

`species`	`"human"`, `"mouse"` or any of the bioconductor or kegg-style abbreviations.
`id.type`	Gene identifiers are returned by the REST service as entrez identifiers. Set this to `"ensembl"` to translate them internally using `convertIdentifiers()`. If `species`is not `"human"` or `"mouse"`, you need to provide an idxref table that works with `convertIdentifiers()`.
`...`	pass through arguments

Details

Currently we just support the pathway database, and only entrez ids.

Note that **it is your responsibility** to ensure that you can use the KEGG database according to their licensing requirements.

Value

A BiocSet of the kegg stuffs

Functions

- `getKeggGeneSetDb()`: method that returns a GeneSetDb

Examples

```
# connects to the internet and takes a while
mouse.entrez <- getKeggCollection("mouse", id.type = "entrez")
human.enrez <- getKeggCollection("human", id.type = "entrez")
```

`getMSigCollection` *Fetches gene set collections from the moleular signature database (MSigDB)*

Description

This provides versioned genesets from gene set collections defined in MSigDB. Collections can be retrieved by their collection name, ie `c("H", "C2", "C7")`.

Usage

```
getMSigCollection(
  collection = NULL,
  species = "human",
  id.type = c("ensembl", "entrez", "symbol", "uniprot"),
  with.kegg = FALSE,
  promote.subcollection = FALSE,
  prefix.collection = FALSE,
  strip.subcollection.prefix = TRUE,
  merge.human.into.mouse = TRUE,
  ...
)

getMSigGeneSetDb(
  collection = NULL,
  species = "human",
  id.type = c("ensembl", "entrez", "symbol", "uniprot"),
  with.kegg = FALSE,
  promote.subcollection = FALSE,
  prefix.collection = FALSE,
  strip.subcollection.prefix = TRUE,
  merge.human.into.mouse = TRUE,
  refetch = FALSE,
  ...
)
```

Arguments

collection	character vector specifying the collections you want (c1, c2, ..., c7, h). By default we load just the hallmark collecitons. Setting this to NULL loads all collections. Alternative you can also include named subsets of collections, like "reactome". Refer to the Details section for more information.
species	"human" or "mouse"? Really, this is anything available in the alias column of the sparrow:::species_info() table (except cyno).
id.type	do you want the feature id's used in the gene sets to be "ensembl" (default), "entrez", or "symbol".
with.kegg	The Broad distributes the latest versions of the KEGG genesets as part of the c2 collection. These genesets come with a restricted license, so by default we do not return them as part of the GeneSetDb. To include the KEGG gene sets when asking for the c2 collection, set this flag to TRUE.
promote.subcollection	there are different sources of genesets for a number of the collections in MSigDB. These are included in the gs_subcollection column of geneSets(this). When this is set to TRUE, the collection column for the genesets is appended with the subcollection. So, instead of having a massive "C2" collection, you'll have bunch of collections like "C2_CGP", "C2_CP:BIOCARTA", etc.
prefix.collection	When TRUE (default: FALSE), the "C1", "C2", etc. is prefixed with "MSigDB_*"
strip.subcollection.prefix	removes the CGP: type prefixes for the gs_subcollection column, except for the C5 GO collection.
merge.human.into.mouse	When TRUE (default), the OG human collections are merged into the newly minted (as of 2024) M* mouse collections. Set to FALSE to not do that.
...	pass through parameters
refetch	If TRUE, this function will requiry the msigdbr package to fetch genesets it has already retrieved and converted. When FALSE, the cached version of the genesets will be returned.

Value

a BiocSet of the MSigDB collections

Functions

- getMSigGeneSetDb(): retrieval method for a GeneSetDb container

Species and Identifier types

This function utilizes the functionality from the {msigdbr} and {babelgene} packages to retrieve gene set definitions from a variety of organisms and identifier types.

KEGG Gene Sets

Due to the licensing restrictions over the KEGG collections, they are not returned from this function unless they are explicitly asked for. You can ask for them through this function by either (i) querying for the `"c2"` collection while setting `with.kegg = TRUE`; or (ii) explicitly calling with `collection = "kegg"`.

Citing the Molecular Signatures Database

To cite your use of the Molecular Signatures Database (MSigDB), please reference Subramanian, Tamayo, et al. (2005, PNAS 102, 15545-15550) and one or more of the following as appropriate:

- Liberzon, et al. (2011, Bionformatics);
- Liberzon, et al. (2015, Cell Systems); and
- The source for the gene set as listed on the gene set page.

Examples

```
# these take a while to load initially, so put them in dontrun blocks.
# you should run these interactively to understand what they return
bcs <- getMSigCollection("h", "human", "entrez")
bcs.h.entrez <- getMSigCollection(c("h", "c2"), "human", "entrez")
bcs.h.ens <- getMSigCollection(c("h", "c2"), "human", "ensembl")
bcs.m.entrez <- getMSigCollection(c("h", "c2"), "mouse", "entrez")

gdb <- getMSigGeneSetDb("h", "human", "entrez")
```

---

getPantherCollection *Get pathways/GOslim collections from PANTHER.db Biocondcutor package.*

Description

This is a convience function that orchestrates the PANTHER.db package to return GeneSetDb objects for either pathway or GOslim information for human or mouse.

Usage

```
getPantherCollection(
  type = c("pathway", "goslim"),
  species = c("human", "mouse")
)

getPantherGeneSetDb(
  type = c("pathway", "goslim"),
  species = c("human", "mouse")
)
```

Arguments

type	"pathway" or, "goslim"
species	"human" or "mouse"

Details

Note that for some reason the PANTHER.db package needs to be installed in a user-writable package location for this to work properly. If you see an error like "Error in resqlite_send_query ... attempt to write a readonly database", this is the problem. Please install another version of the PANTHER.db package in a user-writable directory using BiocManager::install().

Value

A BiocSet of panther pathways

Functions

- getPantherGeneSetDb(): returns a GeneSetDb

GOSLIM

GO Slims are "cut down" versions of the GO ontology that contain a subset of the terms in the whole GO.

PANTHER provides their own set of GO slims, although it's not clear how often these get updated.

Examples

```
# this requires you have the PANTHER.db package installed via BiocManager
bsc.panther <- getPantherCollection(species = "human")
```

---

getReactomeCollection *Retrieve gene set collections from from reactome.db*

Description

Retrieve gene set collections from from reactome.db

Usage

```
getReactomeCollection(
  species = "human",
  id.type = c("entrez", "ensembl"),
  rm.species.prefix = TRUE
)

getReactomeGeneSetDb(
```

```
  species = "human",
  id.type = c("entrez", "ensembl"),
  rm.species.prefix = TRUE
)
```

Arguments

species	the species to get pathay information for
id.type	"entrez" or "ensembl"
rm.species.prefix	pathways are provided with species prefixes from reactome.db, when TRUE (default), these are stripped from the gene set names.

Value

a reactome BiocSet object

Functions

- getReactomeGeneSetDb(): returns a GeneSetDb object

Examples

```
bsc.h <- getReactomeCollection("human")
gdb.h <- getReactomeGeneSetDb("human")
```

---

goseq *Perform goseq Enrichment tests across a GeneSetDb.*

Description

Note that we do not import things from goseq directly, and only load it if this function is fired. I can't figure out a way to selectively import functions from the goseq package without it having to load its dependencies, which take a long time – and I don't want loading sparrow to take a long time. So, the goseq package has moved to Suggests and then is loaded within this function when necessary.

Usage

```
goseq(
  gsd,
  selected,
  universe,
  feature.bias,
  method = c("Wallenius", "Sampling", "Hypergeometric"),
  repcnt = 2000,
  use_genes_without_cat = TRUE,
```

```
  plot.fit = FALSE,
  do.conform = TRUE,
  as.dt = FALSE,
  .pipelined = FALSE
)
```

Arguments

gsd	The GeneSetDb object to run tests against
selected	The ids of the selected features
universe	The ids of the universe
feature.bias	a named vector as long as nrow(x) that has the "bias" information for the features/genes tested (ie. vector of gene lengths). names(feature.bias) should equal rownames(x). If this is not provided, all feature lengths are set to 1 (no bias). The goseq package provides a getlength function which facilitates getting default values for these if you do not have the correct values used in your analysis.
method	The method to use to calculate the unbiased category enrichment scores
repcnt	Number of random samples to be calculated when random sampling is used. Ignored unless method="Sampling".
use_genes_without_cat	A boolean to indicate whether genes without a categorie should still be used. For example, a large number of gene may have no GO term annotated. If this option is set to FALSE, those genes will be ignored in the calculation of p-values (default behaviour). If this option is set to TRUE, then these genes will count towards the total number of genes outside the category being tested.
plot.fit	parameter to pass to goseq::nullp().
do.conform	By default TRUE: does some gymnastics to conform the gsd to the universe vector. This should neber be set to FALSE, but this parameter is here so that when this function is called from the seas() codepath, we do not have to reconform the GeneSetDb object, because it has already been done.
as.dt	If FALSE (default), the data.frame like thing that this funciton returns will be set to a data.frame. Set this to TRUE to keep this object as a data.table
.pipelined	If this is being called external to a seas pipeline, then some additional cleanup of columns name output will be done when FALSE (default). Otherwise the column renaming and post processing is left to the do.goseq caller.

Value

A data.table of results, similar to goseq output. The output from nullp is added to the outgoing data.table as an attribue named "pwf".

References

Young, M. D., Wakefield, M. J., Smyth, G. K., Oshlack, A. (2010). Gene ontology analysis for RNA-seq: accounting for selection bias. *Genome Biology* 11, R14. http://genomebiology.com/2010/11/2/R14

Examples

```
vm <- exampleExpressionSet()
gdb <- conform(exampleGeneSetDb(), vm)

# Identify DGE genes
mg <- seas(vm, gdb, design = vm$design)
lfc <- logFC(mg)

# wire up params
selected <- subset(lfc, significant)$feature_id
universe <- rownames(vm)
mylens <- setNames(vm$genes$size, rownames(vm))
degenes <- setNames(integer(length(universe)), universe)
degenes[selected] <- 1L

gostats <- sparrow::goseq(
  gdb, selected, universe, mylens,
  method = "Wallenius", use_genes_without_cat = TRUE)
```

`gsdScore` *Single sample geneset score using SVD based eigengene value per sample.*

Description

This method was developed by Jason Hackney and first introduced in the following paper doi:10.1038/ng.3520. It produces a single sample gene set score in values that are in "expression space," the innards of which mimic something quite similar to an eigengene based score.

To easily use this method to score a number of gene setes across an experiment, you'll want to have the `scoreSingleSamples()` method drive this function via specifying `"svd"` as one of the `methods`.

Usage

```
gsdScore(
  x,
  eigengene = 1L,
  center = TRUE,
  scale = TRUE,
  uncenter = center,
  unscale = scale,
  retx = FALSE,
  ...,
  .use_irlba = FALSE,
  .drop.sd = 1e-04
)
```

Arguments

x	An expression matrix of genes x samples. When using this to score geneset activity, you want to reduce the rows of x to be only the genes from the given gene set.
eigengene	the "eigengene" you want to get the score for. only accepts a single value for now.
center, scale	center and/or scale data before scoring?
uncenter, unscale	uncenter and unscale the data data on the way out? Defaults to the respective values of center and scale
retx	Works the same as retx from prcomp. If TRUE, will return a retpcax matrix that has the rotated variables.
...	these aren't used in here
.use_irlba	when TRUE, used irlba::svdr() instead of base::svd(). Default: FALSE.
.drop.sd	When zero-sd (non varying) features are scaled, their values are NaN. When the Features with rowSds < this threshold (default 1e-4) are identified, and their scaled values are set to 0.

Details

The difference between this method vs the eigengene score is that the SVD is used to calculate the eigengene. The vector of eigengenes (one score per sample) is then multiplied through by the SVD's left matrix. This produces a matrix which we then take the colSums of to get back to a single sample score for the geneset.

Why do all of that? You get data that is back "in expression space" and we also run around the problem of sign of the eigenvector. The scores you get are very similar to average zscores of the genes per sample, where the average is weighted by the degree to which each gene contributes to the principal component chosen by eigengene, as implemented in the eigenWeightedMean() function.

The core functionality provided here is taken from the soon to be released GSDecon package by Jason Hackney

Value

A list of useful transformation information. The caller is likely most interested in the $score vector, but other bits related to the SVD/PCA decomposition are included for the ride.

Examples

```
vm <- exampleExpressionSet(do.voom=TRUE)
gdb <- conform(exampleGeneSetDb(), vm)
features <- featureIds(gdb, "c2", "BURTON_ADIPOGENESIS_PEAK_AT_2HR")
scores <- gsdScore(vm[features,])$score

## Use scoreSingleSamples to facilitate scoring of all gene sets
scores.all <- scoreSingleSamples(gdb, vm, 'gsd')
s2 <- with(subset(scores.all, name == 'BURTON_ADIPOGENESIS_PEAK_AT_2HR'),
```

```
            setNames(score, sample_id))
all.equal(s2, scores)
```

---

hasGeneSet *Check to see if the GeneSetDb has a collection,name GeneSet defined*

Description

Check to see if the GeneSetDb has a collection,name GeneSet defined

Usage

```
hasGeneSet(x, collection, name, as.error = FALSE)
```

Arguments

x	GeneSetDb
collection	character indicating the collection
name	character indicating the name of the geneset
as.error	If TRUE, a test for the existance of the geneset will throw an error if the geneset does not exist

Value

logical indicating whether or not the geneset is defined.

Examples

```
gdb <- exampleGeneSetDb()
hasGeneSet(gdb, c('c2', 'c7'), c('BIOCARTA_AGPCR_PATHWAY', 'something'))
```

---

hasGeneSetCollection *Check if a collection exists in the* GeneSetDb

---

Description

Check if a collection exists in the GeneSetDb

Usage

```
hasGeneSetCollection(x, collection, as.error = FALSE)
```

Arguments

x	A GeneSetDb()
collection	character vector of name(s) of the collections to query
as.error	logical if TRUE, this will error instead of returning FALSE

Value

logical indicating if this collection exists

Examples

```
gdb <- exampleGeneSetDb()
hasGeneSetCollection(gdb, "c2")
hasGeneSetCollection(gdb, "unknown collection")
```

`incidenceMatrix` *Creates a 1/0 matrix to indicate geneset membership to target object.*

Description

Generates an inidcator matrix to indicate membership of genes (columns) to gene sets (rows). If `y` is provided, then the incidence is mapped across the entire feature-space of `y`.

Usage

```
incidenceMatrix(x, y, ...)
```

Arguments

`x`	A `GeneSetDb()`
`y`	(optional) A target (expression) object `x` is (or can be) conformed to
`...`	parameters passed down into `conform()`.

Value

incidence matrix with nrows = number of genesets and columns are featureIDs. If `y` is passed in, the columns of the returned value match the rows of `y`.

Examples

```
vm <- exampleExpressionSet()
gdb <- exampleGeneSetDb()
im <- incidenceMatrix(gdb)
imv <- incidenceMatrix(gdb, vm)
```

iplot *Visualize gene level behavior of genes within a geneset across a contrast.*

Description

It is informative to look at the individual log fold changes of the genes within a gene set to explore the degree to which they (1) are coherent with respect to each other; and (2) see how the compare to the background distribution of log fold changes of the entire transcriptome.

You can visualize this behavior via a `type = "density"` plot, or a `type = "boxplot". It is also common to plot either = "logFC"or t-statisticsvalue = "t"‘`.

Usage

```
iplot(
  x,
  name,
  value = "logFC",
  type = c("density", "gsea", "boxplot"),
  tools = c("wheel_zoom", "box_select", "reset", "save"),
  main = NULL,
  with.legend = TRUE,
  collection = NULL,
  shiny_source = "mggenes",
  width = NULL,
  height = NULL,
  ggtheme = ggplot2::theme_bw(),
  trim = 0.005,
  ...
)
```

Arguments

x	A SparrowResult() object
name	the name of the geneset to plot
value	A string indicating the column name for the value of the gene-level metadata to plot. Default is "logFC". Anoter often used choice might also be "t", to plot t-statistics (if they're in the result). But this can be any numeric column found in the data.frame returned by geneSet(x, y, j). If this is a named string (vector), then the value in names(value) will be used on the axis when plotted.
type	plot the distributions as a "density" plot or "boxplot".
tools	the tools to display in the rbokeh plot
main	A title to display. If not specified, the gene set name will be used, otherwise you can pass in a custom title, or NULL will disable the title altogether.

with.legend	Draws a legend to map point color to meaning. There are three levels a point (gene level statistic) can be color as, "notsig", "psig", and "sig". "notsig" implies that the FDR >= 10%, "psig" means that FDR <= 10%, but the logFC is "unremarkable" (< 1), and "sig" means that both the FDR <= 10% and the logFC >= 1
collection	If you have genesets with duplicate names in x (only possible with a GeneSetDb object), provide the name of the collection here to disambiguate (default: NULL).
shiny_source	the name of this element that is used in shiny callbacks. Defaults to "mggenes".
width, height	the width and height of the output plotly plot
ggtheme	a ggplot theme, like the thing returned from ggplot2::theme_bw(), for instance.
trim	used to define the upper and lower quantiles to max out the individual gene statistics in the selected geneset.
...	pass through parameters to internal boxplot/density/gsea plotting functions

Value

the ploty plot object

Examples

```
mgr <- exampleSparrowResult()
iplot(mgr, "BURTON_ADIPOGENESIS_PEAK_AT_2HR",
      value = c("t-statistic" = "t"),
      type = "density")
iplot(mgr, "BURTON_ADIPOGENESIS_PEAK_AT_2HR",
      value = c("log2FC" = "logFC"),
      type = "boxplot")
iplot(mgr, "BURTON_ADIPOGENESIS_PEAK_AT_2HR",
      value = c("-statistic" = "t"),
      type = "gsea")
```

---

is.active *Interrogate "active" status of a given geneset.*

Description

Returns the active status of genesets, which are specified by their collection,name compound keys. This function is vectorized and supports query of multiple gene sets at a time. If a requested collection,name gene set doesn't exist, this throws an error.

Usage

```
is.active(x, i, j)
```

Arguments

x	GeneSetDb()
i	collection of geneset(s)
j	name of geneset(s) (must be same length as i.

Value

logical indicating if geneset is active. throws an error if any requested geneset does not exist in x.

Examples

```
dge.stats <- exampleDgeResult()
y <- exampleExpressionSet(do.voom = FALSE)
gdb <- conform(exampleGeneSetDb(), y, min.gs.size = 10)
# size 9 geneset:
geneSet(gdb, "c2", "BYSTRYKH_HEMATOPOIESIS_STEM_CELL_IL3RA")
is.active(gdb, "c2", "BYSTRYKH_HEMATOPOIESIS_STEM_CELL_IL3RA")
# geneset with >100 genes
is.active(gdb, "c7", "GSE3982_MAC_VS_NEUTROPHIL_LPS_STIM_DN")
```

---

logFC *Extract the individual fold changes statistics for elements in the expression object.*

Description

Extract the individual fold changes statistics for elements in the expression object.

Usage

```
logFC(x, as.dt = FALSE)
```

Arguments

x	A SparrowResult()
as.dt	If FALSE (default), the data.frame like thing that this funciton returns will be set to a data.frame. Set this to TRUE to keep this object as a data.table

Value

The log fold change ‘data.table“

Examples

```
vm <- exampleExpressionSet(do.voom=TRUE)
gdb <- exampleGeneSetDb()
mg <- seas(vm, gdb, design = vm$design, contrast = 'tumor')
lfc <- logFC(mg)
```

mgheatmap *Creates a "geneset smart" ComplexHeatmap::Heatmap*

Description

Before we get started, note that you probably want to use `mgheatmap2()`.

This function encapsulates many common "moves" you'll make when trying to make a heatmap, especially if you are trying to show geneset activity across a panel of samples.

NOTE: this function will **almost certainly** reorder the rows of the input matrix. If you are concatentating Heatmap objects together horizontally (ie. you if you want to use a rowAnnotation along side the returned heatmap), you must reorder the rows of the annotation data.frame, ie. `ranno.df <- ranno.df[rownames(out@matrix),]`

Usage

```
mgheatmap(
  x,
  gdb = NULL,
  col = NULL,
  aggregate.by = c("none", "ewm", "ewz", "zscore"),
  split = TRUE,
  scores = NULL,
  gs.order = NULL,
  name = NULL,
  rm.collection.prefix = TRUE,
  rm.dups = FALSE,
  recenter = FALSE,
  rescale = FALSE,
  center = TRUE,
  scale = TRUE,
  rename.rows = NULL,
  zero_center_colramp = NULL,
  zlim = NULL,
  transpose = FALSE,
  ...
)
```

Arguments

x	the data matrix
gdb	GeneSetDb object that holds the genesets to plot. Defaults to NULL, which will plot all rows in x.
col	a colorRamp(2) function
aggregate.by	the method used to generate single-sample geneset scores. Default is none which plots heatmap at the gene level

split — introduce row-segmentation based on genesets or collections? Defaults is TRUE which will create split heatmaps based on collection if aggregate.by != 'none', or based on gene sets if aggregate.by == "none".

scores — If aggregate.by != "none" you can pass in a precomupted scoreSingleSamples() result, otherwise one will be computed internally. Note that if this is a data.frame of pre-computed scores, the gdb is largely irrelevant (but still required).

gs.order — This is experimental, and is here to help order the order of the genesets (or genesets collection) in a different way than the default. By default, gs.order = NULL and genesets are enumerated in alphabetical in the heatmap. You can pass in a character vector that will dictate the order of the genesets displayed in the heatmap. Currently this only matches against the "name" value of the geneset and probably only works when split = TRUE. We will support colleciton,name tuples soon. This can be a superset of the names found in gdb. As of ComplexHeatmap v2 (maybe earlier versions), this doesn't really work when cluster_rows = TRUE.

name — passed down to ComplexHeatmap::Heatmap()

rm.collection.prefix — When TRUE (default), removes the collection name from the genesets annotated on the heatmap.

rm.dups — if aggregate.by == 'none', do we remove genes that appear in more than one geneset? Defaults to FALSE

recenter — do you want to mean center the rows of the heatmap matrix prior to calling ComplexHeatmap::Heatmap()?

rescale — do you want to standardize the row variance to one on the values of the heatmap matrix prior to calling ComplexHeatmap::Heatmap()?

center, scale — boolean parameters passed down into the the single sample gene set scoring methods defined by aggregate.by

rename.rows — defaults to NULL, which induces no action. Specifying a paramter here assumes you want to rename the rows of the heatmap. Please refer to the "Renaming Rows" section for details.

zero_center_colramp — Used to specify the type of color ramp to generate when col is NULL. By default (NULL) we try to guess if we should generate a 0-centered (blue, white, red) color ramp, or an absolute (viridis style) one. The guessing functionality isn't that great, so it doesn't hurt to explicitly set this to TRUE or FALSE.

zlim — Used to control the color saturation of the heatmap when the col parameter is not provided. If NULL, (default), extreme values (outside the c(0.025, 0.975) quantiles) are axed and the colorRamp is based on the remaining value range. If FALSE, the range of the colorRamp is defined by the min/max values. Otherwise a length(2) numeric can be supplied. If the values are between [0,1], then we assume this is a quantile range to be calculated. Otherwise the number are assumed to mark the top and bottom of the color scale range you want to use.

transpose — Flip display so that rows are columns. Default is FALSE.

... — parameters to send down to scoreSingleSamples(), ComplexHeatmap::Heatmap(), renameRows() internal as_matrix().

Details

More info here.

Value

A Heatmap object.

Renaming Heatmap Rows

This function leverages renameRows() so that you can better customize the output of your heatmaps by tweaking its rownames.

If you are plotting a **gene-level** heatmap (ie. aggregate.by == "none"``) and the rownames()are gene identifiers, but you want the rownames of the heatmap to be gene symbols. You can perform parameter.

- If rename.rows is NULL, then nothing is done.
- If rename.rows is a string, then we assume that x has an associated metadata data.frame over its rows and that rename.rows names one of its columns, ie. DGEList$genes[[rename.rows]] or fData(ExpressionSet)[[rename.rows]]. The values in that column will be swapped out for x's rownames
- If rename.rows is a two-column data.frame, the first column is assumed to be rownames(x) and the second is what you want to rename it to.
- When there are duplicates in the renamed rownames, the rename.duplicates ... parameter dictates the behavior. This will happen, for instance, if you are trying to rename the rows of an affy matrix to gene symbols, where we have multiple probe ids for one gene. When rename.duplicates is set to "original", one of the rows will get the new name, and the remaning duplicate rows will keep the rownames they came in with. When set to "make.unique", the new names will contain *.1, *.2, etc. suffixes, as you would get from using base::make.unique().

Maybe you are aggregating the expression scores into geneset scores, and you don't want the rownames of the heatmap to be collection;;name (or just name when rm.collection.prefx = TRUE), you can pass in a two column data.frame, where the first column is collection;name and the second is the name you want to rename that to. There is an example of this in the "Examples" section here.

See Also

mgheatmap2()

Examples

```
library(ComplexHeatmap)
vm <- exampleExpressionSet()
gdb <- exampleGeneSetDb()
col.anno <- ComplexHeatmap::HeatmapAnnotation(
  df = vm$targets[, c("Cancer_Status", "PAM50subtype")],
  col = list(
    Cancer_Status = c(normal = "grey", tumor = "red"),
```

```
    PAM50subtype = c(Basal = "purple", Her2 = "green", LumA = "orange")))
mgh <- mgheatmap(vm, gdb, aggregate.by = "ewm", split=TRUE,
                 top_annotation = col.anno, show_column_names = FALSE,
                 column_title = "Gene Set Activity in BRCA subset")

# Maybe you want the rownames of the matrix to use spaces instead of "_"
rr <- geneSets(gdb)[, "name", drop = FALSE]
rr$newname <- gsub("_", " ", rr$name)
mg2 <- mgheatmap(vm, gdb, aggregate.by='ewm', split=TRUE,
                 top_annotation = col.anno, show_column_names = FALSE,
                 column_title = "Gene Set Activity in BRCA subset",
                 rename.rows = rr)
```

`mgheatmap2` *Creates a "geneset smart" ComplexHeatmap::Heatmap*

Description

Encapsulates many common "moves" you'll make when trying to make a heatmap, especially if you are trying to show geneset activity across a panel of samples.

NOTE: this function will **almost certainly** reorder the rows of the input matrix. If you are concatentating Heatmap objects together horizontally (ie. you if you want to use a rowAnnotation along side the returned heatmap), you must reorder the rows of the annotation data.frame, ie. `ranno.df <- ranno.df[rownames(out@matrix),]`

Usage

```
mgheatmap2(
  x,
  gdb = NULL,
  col = NULL,
  aggregate.by = c("none", "ewm", "ewz", "zscore"),
  split = TRUE,
  scores = NULL,
  gs.order = NULL,
  name = NULL,
  rm.collection.prefix = TRUE,
  rm.dups = FALSE,
  recenter = FALSE,
  rescale = FALSE,
  center = FALSE,
  scale = FALSE,
  uncenter = FALSE,
  unscale = FALSE,
  rename.rows = NULL,
  zlim = NULL,
  transpose = FALSE,
```

```
  ...
)
```

Arguments

x	the data matrix
gdb	GeneSetDb object that holds the genesets to plot. Defaults to NULL, which will plot all rows in x.
col	a colorRamp(2) function
aggregate.by	the method used to generate single-sample geneset scores. Default is none which plots heatmap at the gene level
split	introduce row-segmentation based on genesets or collections? Defaults is TRUE which will create split heatmaps based on collection if aggregate.by != 'none', or based on gene sets if aggregate.by == "none".
scores	If aggregate.by != "none" you can pass in a precomupted scoreSingleSamples() result, otherwise one will be computed internally. Note that if this is a data.frame of pre-computed scores, the gdb is largely irrelevant (but still required).
gs.order	This is experimental, and is here to help order the order of the genesets (or genesets collection) in a different way than the default. By default, gs.order = NULL and genesets are enumerated in alphabetical in the heatmap. You can pass in a character vector that will dictate the order of the genesets displayed in the heatmap. Currently this only matches against the "name" value of the geneset and probably only works when split = TRUE. We will support colleciton,name tuples soon. This can be a superset of the names found in gdb. As of ComplexHeatmap v2 (maybe earlier versions), this doesn't really work when cluster_rows = TRUE.
name	passed down to ComplexHeatmap::Heatmap()
rm.collection.prefix	When TRUE (default), removes the collection name from the genesets annotated on the heatmap.
rm.dups	if aggregate.by == 'none', do we remove genes that appear in more than one geneset? Defaults to FALSE
recenter	do you want to mean center the rows of the heatmap matrix prior to calling ComplexHeatmap::Heatmap()? This is passed down to scale_rows(). Look there for more mojo.
rescale	do you want to standardize the row variance to one on the values of the heatmap matrix prior to calling ComplexHeatmap::Heatmap()? This is passed down to scale_rows(). Look there for more mojo.
center, scale, uncenter, unscale	boolean parameters passed down into the the single sample gene set scoring methods defined by aggregate.by
rename.rows	defaults to NULL, which induces no action. Specifying a paramter here assumes you want to rename the rows of the heatmap. Please refer to the "Renaming Rows" section for details.

zlim	A length(zlim) == 2 numeric vector that defines the min and max values from x for the circlize::colorRamp2 call. If the heatmap that is being drawn is "0-centered"-ish, then this defines the real values of the fenceposts. If not, then these define the quantiles to trim off the top or bottom.
transpose	Flip display so that rows are columns. Default is FALSE.
...	parameters to send down to scoreSingleSamples(), ComplexHeatmap::Heatmap(), renameRows() internal as_matrix().

Details

More info here.

Value

A Heatmap object.

Renaming Heatmap Rows

This function leverages renameRows() so that you can better customize the output of your heatmaps by tweaking its rownames.

If you are plotting a **gene-level** heatmap (ie. aggregate.by == "none"``) and the rownames()are gene identifiers, but you want the rownames of the heatmap to be gene symbols. You can perform parameter.

- If rename.rows is NULL, then nothing is done.
- If rename.rows is a string, then we assume that x has an associated metadata data.frame over its rows and that rename.rows names one of its columns, ie. DGEList$genes[[rename.rows]] or fData(ExpressionSet)[[rename.rows]]. The values in that column will be swapped out for x's rownames
- If rename.rows is a two-column data.frame, the first column is assumed to be rownames(x) and the second is what you want to rename it to.
- When there are duplicates in the renamed rownames, the rename.duplicates ... parameter dictates the behavior. This will happen, for instance, if you are trying to rename the rows of an affy matrix to gene symbols, where we have multiple probe ids for one gene. When rename.duplicates is set to "original", one of the rows will get the new name, and the remaning duplicate rows will keep the rownames they came in with. When set to "make.unique", the new names will contain *.1, *.2, etc. suffixes, as you would get from using base::make.unique().

Maybe you are aggregating the expression scores into geneset scores, and you don't want the rownames of the heatmap to be collection;;name (or just name when rm.collection.prefx = TRUE), you can pass in a two column data.frame, where the first column is collection;name and the second is the name you want to rename that to. There is an example of this in the "Examples" section here.

Examples

```
vm <- exampleExpressionSet()
gdb <- exampleGeneSetDb()
col.anno <- ComplexHeatmap::HeatmapAnnotation(
  df = vm$targets[, c("Cancer_Status", "PAM50subtype")],
  col = list(
    Cancer_Status = c(normal = "grey", tumor = "red"),
    PAM50subtype = c(Basal = "purple", Her2 = "green", LumA = "orange")))
mgh <- mgheatmap2(vm, gdb, aggregate.by = "ewm", split = TRUE,
                  top_annotation = col.anno, show_column_names = FALSE,
                  column_title = "Gene Set Activity in BRCA subset")
ComplexHeatmap::draw(mgh)

# Center to "normal" group
mgc <- mgheatmap2(vm, gdb, aggregate.by = "ewm", split = TRUE,
                  top_annotation = col.anno, show_column_names = FALSE,
                  recenter = vm$targets$Cancer_Status == "normal",
                  column_title = "Gene Set Activity in BRCA subset")
ComplexHeatmap::draw(mgc)
# Maybe you want the rownames of the matrix to use spaces instead of "_"
rr <- geneSets(gdb)[, "name", drop = FALSE]
rr$newname <- gsub("_", " ", rr$name)
mg2 <- mgheatmap2(vm, gdb, aggregate.by='ewm', split=TRUE,
                  top_annotation = col.anno, show_column_names = FALSE,
                  column_title = "Gene Set Activity in BRCA subset",
                  rename.rows = rr)
```

`msg` *Utility function to cat a message to stderr (by default)*

Description

Utility function to cat a message to stderr (by default)

Usage

```
msg(..., file = stderr())
```

Arguments

`...`	pieces of the message
`file`	where to send the message. Defaults to `stderr()`

Value

Nothing, dumps text to `file`

Examples

```
msg("this is a message", "to stderr")
```

`ora` *Performs an overrepresentation analysis, (optionally) accounting for bias.*

Description

This function wraps `limma::kegga()` to perform biased overrepresntation analysis over gene set collection stored in a GeneSetDb (`gsd`) object. Its easiest to use this function when the biases and selection criteria are stored as columns of the input data.frame `dat`.

Usage

```
ora(
  x,
  gsd,
  selected = "significant",
  groups = NULL,
  feature.bias = NULL,
  universe = NULL,
  restrict.universe = FALSE,
  plot.bias = FALSE,
  ...,
  as.dt = FALSE
)

plot_ora_bias(x, selected, feature.bias, ...)
```

Arguments

`x`	A data.frame with feature-level statistics. Minimally, this should have a `"feature_id"` (character) column, but read on ...
`gsd`	The GeneSetDb
`selected`	Either the name of a logical column in `dat` used to subset out the features to run the enrichement over, or a character vector of `"feature_id"`s that are selected from `dat[["feature_id"]]`.
`groups`	Encodes groups of features that we can use to test selected features individual, as well as "all" together. This can be specified by: (1) specifying a name of a column in `dat` to split the enriched features into subgroups. (2) A named list of features to intersect with `selected`. By default this is `NULL`, so we only run enrichment over all elements in `selected`. See examples for details.
`feature.bias`	If `NULL` (default), no bias is used in enrichment analysis. Otherwise, can be the name of a column in `dat` to extract a numeric bias vector (gene length, GC content, average expression, etc.) or a named (using featureIds) numeric vector of the same. The BiasedUrn CRAN package is required when this is not NULL.

universe	Defaults to all elements in dat[["feature_id"]].
restrict.universe	See same parameter in limma::kegga()
plot.bias	See plot parameter in limma::kegga(). You can generate this plot without running ora using the plot_ora_bias(), like so: plot_ora_bias(dat, selected = selected, groups = groups, feature.bias = feature.bias)
...	parameters passed to conform()
as.dt	If FALSE (default), the data.frame like thing that this funciton returns will be set to a data.frame. Set this to TRUE to keep this object as a data.table

Details

In principle, this test does what goseq does, however I found that sometimes calling goseq would throw errors within goseq::nullp() when calling makesplines. I stumbled onto this implementation when googling for these errors and landing here: https://support.bioconductor.org/p/65789/#65914

The meat and potatoes of this function's code was extracted from limma::kegga(), written by Gordon Smyth and Yifang Hu.

Note that the BiasedUrn CRAN package needs to be installed to support biased enrichment testing

Value

A data.frame of pathway enrichment. The last N colums are enrichment statistics per pathway, grouped by the groups parameter. P.all are the stats for all selected features, and the remaingin P.* columns are for the features specifed by groups.

Functions

- plot_ora_bias(): plots the bias of coviarate to DE / selected status. Code taken from limma::kegga()

References

Young, M. D., Wakefield, M. J., Smyth, G. K., Oshlack, A. (2010). Gene ontology analysis for RNA-seq: accounting for selection bias. *Genome Biology* 11, R14. http://genomebiology.com/2010/11/2/R14

Examples

```
dgestats <- exampleDgeResult()
gdb <- randomGeneSetDb(dgestats)

# Run enrichmnent without accounting for any bias
nobias <- ora(dgestats, gdb, selected = "selected", groups = "direction",
              feature.bias = NULL)

# Run enrichment and account for gene length
lbias <- ora(dgestats, gdb, selected = "selected",
             feature.bias = "effective_length")

# plot length bias with DGE status
```

```
plot_ora_bias(dgestats, "selected", "effective_length")

# induce length bias and see what is the what ..............................
biased <- dgestats[order(dgestats$pval),]
biased$effective_length <- sort(biased$effective_length, decreasing = TRUE)
plot_ora_bias(biased, "selected", "effective_length")
etest <- ora(biased, gdb, selected = "selected",
             groups = "direction",
             feature.bias = "effective_length")
```

`p.matrix` *Assembles a matrix of nominal or adjusted pvalues from a sparrow::seas result*

Description

You might want a matrix of pvalues (or FDRs) for the gene sets across all GSEA methods you tried. I think I did, once, so here it is.

Usage

```
p.matrix(
  x,
  names = resultNames(x),
  pcol = c("padj", "padj.by.collection", "pval")
)
```

Arguments

`x`	A `SparrowResult()` object.
`names`	the entries from `resultNames(x)` that you want to include in the matrix. By default we take all of them.
`pcol`	The name of the column in `logFC(x)` where the type of pvalues are that we are collection. Pick on of `"padj"`, `"padj.by.collection"`, or `"pval"`

Value

A matrix of the desired pvalues for all genesets

Examples

```
mg <- exampleSparrowResult()
pm <- p.matrix(mg)
```

randomGeneSetDb *Generates a fake GeneSetDb by sampling from features in a seas input.*

Description

I wrote this because initial fetching from msigdbr can be slow, and also having some weird crashes in the unit tests of bioc3.14-devel.

Usage

```
randomGeneSetDb(x, n = 10, bias = NULL, include_spaces = TRUE, ...)
```

Arguments

x	an input container to seas()
n	number of genesets
bias	column in x to bias the geneset creation by
include_spaces	If TRUE (default), geneset names will have a space in them. This is used to test corner cases around the assumption of geneset naming conventions.
...	pass through args

Details

This is a helper function for development, and shouldn't be used by normal users of this package.

Value

A randomly generated GeneSetDb you can use against x for testing.

Examples

```
gdb.rando <- randomGeneSetDb(exampleDgeResult(), 10, bias = "t")
```

renameCollections *Rename the collections in a GeneSetDb*

Description

This function remaps names of collections in the database from their current names to ones specified by the user, folows the dplyr::rename convenction where names() of the rename vector are the new names you want, and its values are the old names it came from.

Usage

```
renameCollections(x, rename = NULL, ...)
```

Arguments

x	A GeneSetDb object
rename	a named character vector. names(rename) are the names of the collection you want to rename, and their values are the new names.
...	pass it along

Value

GeneSetDb x with renamed geneSets(x)$collection values.

Examples

```
gdb <- exampleGeneSetDb()
ngdb <- renameCollections(gdb, c("MSigDB C2" = "c2", "ImmuneSigDb" = "c7"))
all.equal(
  unname(geneSetURL(gdb, "c7", "GSE3982_BCELL_VS_TH2_DN")),
  unname(geneSetURL(ngdb, "ImmuneSigDb", "GSE3982_BCELL_VS_TH2_DN")))
```

---

renameRows *Smartly/easily rename the rows of an object.*

Description

The most common usecase for this is when you have a SummarizedExperiment, DGEList, matrix, etc. that is "rownamed" by some gene idnetifiers (ensembl, entrez, etc) that you want to "easily" convert to be rownamed by symbols. And perhaps the most common use-case for this, again, would be able to easily change rownames of a heatmap to symbols.

Usage

```
renameRows(x, xref, duplicate.policy = "original", ...)
```

Arguments

x	an object to whose rows need renaming
xref	an object to help with the renaming. • A character vector where length(xref) == nrow(x). Every row in x should correspond to the renamed value in the same position in xref • If x is a DGEList, SummarizedExperiment, etc. this can be a string. In this case, the string must name a column in the data container's fData-like data.frame. The values in that column will be the new candidate rownames for the object. • A two column data.frame. The first column has entries in rownames(x), and the second column is the value to rename it to.

duplicate.policy
: The policy used to deal with duplicates in the renamed values. If Multiple elements in the source can be renamed to the same elements in the target (think of microarray probes to gene symbols), what to do? By deafult (`"original"`), one of the original elements will be renamed to the new name, and the rest will keep their original (unique) names. When set to `"make.unique"`, the new name will be kept, but `*.1`, `*.2`, etc. will be appended to all but the first multimapper.

...
: pass through variable down to default method

Details

The rownames that can't successfully remapped will keep their old names. This function should also guarantee that the rows of the incoming matrix are the same as the outgoing one.

Value

An updated version of `x` with freshly minted rownames.

Examples

```
eset <- exampleExpressionSet(do.voom = FALSE)
ess <- renameRows(eset, "symbol")

vm <- exampleExpressionSet(do.voom = TRUE)
vms <- renameRows(vm, "symbol")
```

---

resultNames *Interrogate the results of a sparrow::seas analysis stored in a SparrowResult*

Description

The `resultNames`, `result`, and `results` functions enable you to explore the results of the analysis run with `seas()`.

The results that are stored within a `SparrowResult` object have a more or less 1:1 mapping with the values passed as `methods`, parameter of the `seas()` call.

Generates a table to indicate the number of genesets per collection that pass a given FDR. The table provides separate groups of rows for each of the `methods` run in the `seas()` call that generated that generated `x`.

Usage

```
resultNames(x)

result(x, ...)

## S3 method for class 'SparrowResult'
```

```
result(
  x,
  name = NULL,
  stats.only = FALSE,
  rank.by = c("pval", "t", "logFC"),
  add.suffix = FALSE,
  as.dt = FALSE,
  ...
)

results(
  x,
  names = resultNames(x),
  stats.only = TRUE,
  rank.by = c("pval", "logFC", "t"),
  add.suffix = length(names) > 1L,
  as.dt = FALSE
)

tabulateResults(
  x,
  names = resultNames(x),
  max.p = 0.2,
  p.col = c("padj", "padj.by.collection", "pval"),
  as.dt = FALSE
)
```

Arguments

x	A SparrowResult() object.
...	pass through arguments
name	the names of the results desired
stats.only	logical, set to FALSE if you want to return all (column-wise) data for each result. By default only the pvalues, adjusted pvalues, and rank are returned.
rank.by	the statistic to use to append a rank column for the geneset result. By default we rank by pvalue calculated by the GSEA method. You can rank the results based on the trimmed mean of the logFC's calculated for all of the features in the geneset ("logFC"), or the trimmed t-statistics of the these features ("t").
add.suffix	If TRUE, adds .name as a suffix to the columns of the method-specific statistics returned, ie. the pval column from the "camera" result will be turned to pval.camera.
as.dt	If FALSE (default), the data.frame like thing that this funciton returns will be set to a data.frame. Set this to TRUE to keep this object as a data.table
names	the names of the GSEA methods to be reported. By default, this function will display results for all methods.
max.p	The maximum padj value to consider a result significant
p.col	use padj or padj.by.collection?

Details

The product of an indivdual GSEA is consumed by the corresponding `do.<METHOD>` function and converted into a data.table of results that is internally stored.

Use the `resultNames()` function to identify which results are available for interrogation. The `result()` function returns the statistics of one individual result, and the `results()` function combines the results from the specified methods into an arbitrarily wide data.table with `method`-suffixed column names.

Use the `tabulateResults()` function to create a summary table that tallies the number of significant genesets per collection, per method at the specified FDR thresholds.

Value

a data.table with the results from the requested method.

a data.table that summarizes the significant results per method per collection for the GSEA that was run

Examples

```
res <- exampleSparrowResult()
resultNames(res)
head(result(res, "camera"))
head(results(res))
```

---

scale_rows *Centers and scales the rows of a numeric matrix.*

Description

This was for two reasons: (1) to avoid the (more commonly used) `t(scale(t(x), ...)` idiom; and (2) to specify what values, columns of x, etc. to use to calculate means and sd's to use in the scaling function.

Usage

```
scale_rows(x, center = TRUE, scale = TRUE, ...)
```

Arguments

`x`	the matrix-like object
`center`	Either a logical, character, or numeric-like value that specifies what to center
`scale`	Either a logical, characeter, or numeric-like value that specifies what to scale
`...`	pass through arguments

Details

For instance, you might want to subtract the mean of a subset of columns from each row in the matrix (like the columns that come from control samples)

Note that this method returns different attrs() for scaling and center than base:scale does. Our values are always named.

Value

a scaled version of `x`

Transformation based on specific columns

`center` and `scale` can be a logical, character, or numeric-like vector. The flexibility enables the following scenarios:

1. The user can set it to `TRUE` to center all values on the mean of their row. (`FALSE` does no centering)
2. A (named) vector of values that is a superset of rownames(x). These will be the values that are subtracted from each row.
3. A logical vector as long as ncol(x). Each value will be centered to the mean of the values of the columns specified as TRUE.
4. An integer vector, the is the analog of 3 but specifies the columns to use for centering.

Examples

```
# see tests/testthat/test-scale_rows.R for more examples
m <- matrix(rnorm(50, mean = 1, sd = 2), nrow = 5,
            dimnames = list(LETTERS[1:5], letters[1:10]))
s0 <- scale_rows(m, center = TRUE, scale = FALSE)
all.equal(s0, t(scale(t(m), center = TRUE, scale = FALSE)))

# mean center rows to a specific group of control samples (columns)
ctrl <- sample(colnames(m), 3)
s.ctrl <- scale_rows(m, center = ctrl, scale = FALSE)
ctrl.means <- Matrix::rowMeans(m[, ctrl])
all.equal(s.ctrl, t(scale(t(m), center = ctrl.means, scale = FALSE)))
```

---

scoreSingleSamples *Generates single sample gene set scores across a datasets by many methods*

Description

It is common to assess the activity of a gene set in a given sample. There are many ways to do that, and this method is analogous to the `seas()` function in that it enables the user to run a multitude of single-sample-gene-set-scoring algorithms over a target expression matrix using a `GeneSetDb()` object.

Usage

```
scoreSingleSamples(
  gdb,
  y,
  methods = "ewm",
  as.matrix = FALSE,
  drop.sd = 1e-04,
  drop.unconformed = FALSE,
  verbose = FALSE,
  recenter = FALSE,
  rescale = FALSE,
  ...,
  as.dt = FALSE
)
```

Arguments

gdb	A GeneSetDb
y	An expression matrix to score genesets against
methods	A character vector that enumerates the scoring methods you want to run over the samples. Please reference the "Single Sample Scoring Methods" section for more information.
as.matrix	Return results as a list of matrices instead of a melted data.frame? Defaults to FALSE.
drop.sd	Genes with a standard deviation across columns in y that is less than this value will be dropped.
drop.unconformed	When TRUE, genes in y that are not found in gdb are removed from the expression container. You may want to set this to TRUE when y is very large until better sparse matrix support is injected. This will change the scores for gsva and ssGSEA, though. Default is FALSE.
verbose	make some noise? Defaults to FALSE.
recenter, rescale	If TRUE, the scores computed by each method are centered and scaled using the scale function. These variables correspond to the center and scale parameters in the scale function. Defaults to FALSE.
...	these parameters are passed down into the the individual single sample scoring funcitons to customize them further.
as.dt	If FALSE (default), the data.frame like thing that this funciton returns will be set to a data.frame. Set this to TRUE to keep this object as a data.table

Details

Please refer to the "Generating Single Sample Gene Set Scores" of the sparrow vignette for further exposition.

Value

A long data.frame with sample_id,method,score values per row. If `as.matrix=TRUE`, a matrix with as many rows as `geneSets(gdb)` and as many columns as `ncol(x)`

Single Sample Scoring Methods

The following `methods` are currenly provided.

- `"ewm"`: The `eigenWeightedMean()` calculates the fraction each gene contributes to a pre-specified principal component. These contributions act as weights over each gene, which are then used in a simple weighted mean calculation over all the genes in the geneset per sample. This is similar, in spirit, to the svd/gsdecon method (ie. `method = "gsd"``) You can use this method to perform a` scaleanduncentertoFALSE. "ewz": `with` unscaleanduncenterset toFALSE‘.
- `"gsd"`: This method was first introduced by Jason Hackney in doi:10.1038/ng.3520. Please refer to the `gsdScore()` function for more information.
- `"ssgsea"`: Using ssGSEA as implemented in the GSVA package.
- `"zscore"`: The features in the expression matrix are rowwise z transformed. The gene set level score is then calculated by adding up the zscores for the genes in the gene set, then dividing that number by either the the size (or its sqaure root (default)) of the gene set.
- `"mean"`: Simply take the mean of the values from the expression matrix that are in the gene set. Right or wrong, sometimes you just want the mean without transforming the data.
- `"gsva"`: The gsva method of GSVA package.
- `"plage"`: Using "plage" as implemented in the GSVA package

Examples

```
gdb <- exampleGeneSetDb()
vm <- exampleExpressionSet()
scores <- scoreSingleSamples(
  gdb, vm, methods = c("ewm", "gsva", "zscore"),
  center = TRUE, scale = TRUE, ssgsea.norm = TRUE, as.dt = TRUE)

sw <- data.table::dcast(scores, name + sample_id ~ method, value.var='score')


corplot(
  sw[, c("ewm", "gsva", "zscore")],
  title = "Single Sample Score Comparison")


zs <- scoreSingleSamples(
  gdb, vm, methods = c('ewm', 'ewz', 'zscore'), summary = "mean",
  center = TRUE, scale = TRUE, uncenter = FALSE, unscale = FALSE,
  as.dt = TRUE)
zw <- data.table::dcast(zs, name + sample_id ~ method, value.var='score')


  corplot(zw[, c("ewm", "ewz", "zscore")], title = "EW zscores")
```

seas *Performs a plethora of set enrichment analyses over varied inputs.*

Description

This is a wrapper function that delegates GSEA analyses to different "workers", each of which implements the flavor of GSEA of your choosing. The particular analyses that are performed are specified by the `methods` argument, and these methods are fine tuned by passing their arguments down through the `...` of this wrapper function.

Usage

```
seas(
  x,
  gsd,
  methods = NULL,
  design = NULL,
  contrast = NULL,
  use.treat = FALSE,
  feature.min.logFC = if (use.treat) log2(1.25) else 1,
  feature.max.padj = 0.1,
  trim = 0.1,
  verbose = FALSE,
  ...,
  score.by = c("t", "logFC", "pval"),
  rank_by = NULL,
  rank_order = c("ordered", "descending", "ascending"),
  xmeta. = NULL,
  BPPARAM = BiocParallel::SerialParam()
)
```

Arguments

`x`	An object to run enrichment analyses over. This can be an ExpressoinSet-like object that you can differential expression over (for roast, fry, camera), a named (by feature_id) vector of scores to run ranked-based GSEA, a data.frame with feature_id's, ranks, scores, etc.
`gsd`	The `GeneSetDb()` that defines the gene sets of interest.
`methods`	A character vector indicating the GSEA methods you want to run. Refer to the GSEA Methods section for more details. If no methods are specified, only differential gene expression and geneset level statistics for the contrast are computed.
`design`	A design matrix for the study
`contrast`	The contrast of interest to analyze. This can be a column name of `design`, or a contrast vector which performs "coefficient arithmetic" over the columns of `design`. The `design` and `contrast` parameters are interpreted in *exactly* the

same way as the same parameters in limma's `limma::camera()` and `limma::roast()` methods.

`use.treat` should we use limma/edgeR's "treat" functionality for the gene-level differential expression analysis?

`feature.min.logFC` The minimum logFC required for an individual feature (not geneset) to be considered differentialy expressed. Used in conjunction with `feature.max.padj` primarily for summarization of genesets (by `geneSetsStats()`, but can also be used by GSEA methods that require differential expression calls at the individual feature level, like `goseq()`.

`feature.max.padj` The maximum adjusted pvalue used to consider an individual feature (not geneset) to be differentially expressed. Used in conjunction with `feature.min.logFC`.

`trim` The amount to trim when calculated trimmed `t` and `logFC` statistics for each geneset. This is passed down to the `geneSetsStats()` function.

`verbose` make some noise during execution?

`...` The arguments are passed down into `calculateIndividualLogFC()` and the various geneset analysis functions.

`score.by` This tells us how to rank the features after differential expression analysis when `x` is an expression container. It specifies the name of the column to use downstream of a differential expression analysis over `x`. If `x` is a data.frame that needs to be ranked, see `rank_by`.

`rank_by` Only works when `x` is a data.frame-like input. The name of a column that should be used to rank the features in `x` for pre-ranked gsea tests like cameraPR or fgsea. `rank_by` overrides `score.by`

`rank_order` Only used when `x` is a data.frame-like input. Specifies how the features in `x` should be used to rank the features in `x` using the `rank_by` column. Accepted values are: `"ordered"` (default) means that the rows in `x` are pre-ranked already. `"descendeing"`, and `"ascending"`.

`xmeta.` A hack to support data.frame inputs for `x`. End users should not use this.

`BPPARAM` a *BiocParallel* parameter definition, like one generated from `BiocParallel::MulticoreParam()`, or `BiocParallel::BatchtoolsParam()`, for instance, which is passed down to BiocParallel::bplapply()]. Default is set to `BiocParallel::SerialParam()`

Details

Set enrichment analyses can either be performed over an expression object, which requires the specification of the experiment design and contrast of interest, or over a set of features to rank (stored as a data.frame or vector).

Note that we are currently in the middle of a refactor to accept and fully take advantage of `data.frame` as inputs for `x`, which will be used for preranked type of GSEA methods. See the following issue for more details: https://github.com/lianos/multiGSEA/issues/24

The bulk of the GSEA methods currently available in this package come from edgeR/limma, however others are included (and are being added), as well. *GSEA Methods* and *GSEA Method Parameterization* sections for more details.

In addition to performing GSEA, this function also internally orchestrates a differential expression analysis, which can be tweaked by identifying the parameters in the `calculateIndividualLogFC()` function, and passing them down through `...` here. The results of the differential expression analysis (ie. the `limma::topTable()`) are accessible by calling the `logFC()` function on the `SparrowResult()` object returned from this function call.

Please Note: be sure to cite the original GSEA method when using results generated from this function.

Value

A `SparrowResult()` which holds the results of all the analyses specified in the `methods` parameter.

GSEA Methods

You can choose the methods you would like to run by providing a character vector of GSEA method names to the `methods` parameter. Valid methods you can select from include:

- `"camera"`: from `limma::camera()` (*)
- `"cameraPR"`: from `limma::cameraPR()`
- `"ora"`: overrepresentation analysis optionally accounting for bias (`ora()`). This method is a wrapper function that makes the functionality in `limma::kegga()` available to an arbitrary GeneSetDb.
- `"roast"`: from `limma::roast()` (*)
- `"fry"`: from `limma::fry()` (*)
- `"romer"`: from `limma::romer()` (*)
- `"geneSetTest"`: from `limma::geneSetTest()`
- `"goseq"`: from `goseq::goseq()`
- `"fgsea"`: from `fgsea::fgsea()`

Methods annotated with a `(*)` indicate that these methods require a complete expression object, a valid design matrix, and a contrast specification in order to run. These are all of the same things you need to provide when performing a vanilla differential gene expression analysis.

Methods missing a `(*)` can be run on a feature-named input vector of gene level statistics which will be used for ranking (ie. a named vector of logFC's or t-statistics for genes). They can also be run by providing an expression, design, and contrast vector, and the appropriate statistics vector will be generated internally from the t-statistics, p-values, or log-fold-changes, depending on the value provided in the `score.by` parameter.

The worker functions that execute these GSEA methods are functions named `do.METHOD` within this package. These functions are not meant to be executed directly by the user, and are therefore not exported. Look at the respective method's help page (ie. if you are running `"camera"`, look at the `limma::camera()` help page for full details. The formal parameters that these methods take can be passed to them via the `...` in this `seas()` function.

GSEA Method Parameterization

Each GSEA method can be tweaked via a custom set of parameters. We leave the documentation of these parameters and how they affect their respective GSEA methods to the documentation available in the packages where they are defined. The `seas()` call simply has to pass these parameters down into the `...` parameters here. The `seas` function will then pass these along to their worker functions.

What happens when two different GSEA methods have parameters with the same name?

Unfortunately you currently cannot provide different values for these parameters. An upcoming version version of sparrow will support this feature via slightly different calling semantics. This will also allow the caller to call the same GSEA method with different parameterizations so that even these can be compared against each other.

Differential Gene Expression

When the `seas()` call is given an expression matrix, design, and contrast, it will also internally orchestrate a gene level differential expression analysis. Depending on the type of expression object passed in via `x`, this function will guess on the best method to use for this analysis.

If `x` is a `DGEList`, then ensure that you have already called `edgeR::estimateDisp()` on `x` and edgeR's quasilikelihood framework will be used, otherwise we'll use limma (note that `x` can be an `EList` run through `voom()`, `voomWithQuailityWeights()`, or when where you have leveraged limma's `limma::duplicateCorrelation()` functionality, even.

The parameters of this differential expression analysis can also be customized. Please refer to the `calculateIndividualLogFC()` function for more information. The `use.treat`, `feature.min.logFC`, `feature.max.padj`, as well as the `...` parameters from this function are passed down to that funciton.

Examples

```
vm <- exampleExpressionSet()
gdb <- exampleGeneSetDb()
mg <- seas(vm, gdb, c('camera', 'fry'),
           design = vm$design, contrast = 'tumor',
           # customzie camera parameter:
           inter.gene.cor = 0.04)
resultNames(mg)
res.camera <- result(mg, 'camera')
res.fry <- result(mg, 'fry')
res.all <- results(mg)
```

SparrowResult-class *A SparrowResult object holds the results from a sparrow::seas() call.*

Description

A call to `seas()` will produce analyses for an arbitrary number of GSEA methods, the results of which will be stored and accessible here using the `result()`, `results()`, and `resultNames()`.

In addition, the GeneSetDb() used for the analysis is accessible via geneSetDb(), and the results from the differential expression analysis is available via logFC().

Visualizing results of a geneset based analysis also are functions that operate over a SparrowResult object, for instance see the iplot() and the sparrow.shiny package.

Slots

gsd The GeneSetDb() this analysis was run over

results The list of individual results generated by each of the GSEA methods that were run.

logFC The differential expression statistics for each individual feature measured in the experiment.

---

sparrow_methods *Lists the supported GSEA methods by sparrow*

Description

Lists the supported GSEA methods by sparrow

Usage

```
sparrow_methods()
```

Value

a character vector of GSEA names, or a list of metadata for each method.

Examples

```
sparrow_methods()
```

---

species_info *Match a species query to the regularized species info.*

Description

Match a species query to the regularized species info.

Usage

```
species_info(query = NULL, ...)
```

Arguments

query	the species name to lookup, if NULL (default), returns the internal species info table, otherwise the row of the table that matches query.
...	pass through

Value

a data.frame of species-related information that is used to fetch appropriate annotation files and conversion functions between species for gene identifiers, and such.

Examples

```
species_info()
species_info("human")
```

`ssGSEA.normalize` *Normalize a vector of ssGSEA scores in the ssGSEA way.*

Description

ssGSEA normalization (as implemented in GSVA (ssgsea.norm)) normalizes the individual scores based on ALL scores calculated across samples AND genesets. It does NOTE normalize the scores within each geneset independantly of the others.

Usage

```
ssGSEA.normalize(x, bounds = range(x))
```

Arguments

`x`	a `numeric` vector of ssGSEA scores for a single signature
`bounds`	the maximum and minimum scores obvserved used to normalize against.

Details

This method is an internal utilit function and not exported on purpose

Value

normalized `numeric` vector of `x`

subset.GeneSetDb *Subset GeneSetDb to only include specified genesets.*

Description

This is a utility function that is called by `[.GeneSetDb` and is not exported because it is not meant for external use.

Usage

```
## S3 method for class 'GeneSetDb'
subset(x, keep)
```

Arguments

x	a GeneSetDb()
keep	logical vector as long as `nrow(geneSets(x, active.only=FALSE)`

Details

DEBUG: If keep is all FALSE, this will explode. What does an empty GeneSetDb look like, anyway? Something ...

We want to support a better, more fluent subsetting of GeneSetDb objects. See Issue #12 (https://github.com/lianos/multiGSE

Value

a GeneSetDb that has only the results for the specified genesets.

Examples

```
gdb.all <- exampleGeneSetDb()
gs <- geneSets(gdb.all)
gdb <- gdb.all[gs$collection %in% c("c2", "c7")]
```

subsetByFeatures *Subset a GeneSetDb to only include geneSets with specified features.*

Description

Subset a GeneSetDb to only include geneSets with specified features.

Usage

```
subsetByFeatures(x, features, value = c("feature_id", "x.id", "x.idx"), ...)

## S4 method for signature 'GeneSetDb'
subsetByFeatures(x, features, value = c("feature_id", "x.id", "x.idx"), ...)
```

Arguments

x	GeneSetDb
features	Character vector of featureIds
value	are you feature id's entered as themselves (feature_id), which is the default, or are you querying by their index into a target expression object? This is only relevant if you are working with a conform-ed GeneSetDb, and further you as a user won't likely invoke this argument, but is used internally.
...	pass through arguments

Value

A subset of x which contains only the geneSets that contain features found in featureIds

Methods (by class)

- subsetByFeatures(GeneSetDb): subset GeneSetDb by feature id's

Examples

```
gdb <- exampleGeneSetDb()
features <- c("55839", "8522", "29087")
(gdb.sub <- subsetByFeatures(gdb, features))
```

---

validateInputs *Validate the input objects to a GSEA call.*

Description

Checks to ensure that the values for x, design, and contrast are appropriate for the GSEA methods being used. If they are kosher, then "normalized" versions of these objects are returned in an (aptly) named list, otheerwise an error is thrown.

Usage

```
validateInputs(
  x,
  design = NULL,
  contrast = NULL,
  methods = NULL,
  xmeta. = NULL,
  require.x.rownames = TRUE,
  ...
)
```

Arguments

x	The expression object to use
design	A design matrix, if the GSEA method(s) require it
contrast	A contrast vector (if the GSEA method(s) require it)
methods	A character vector of the GSEA methods that these inputs will be used for.
xmeta.	hack for supportin data.frame inputs.
require.x.rownames	Leave this alone, should always be TRUE but have it in this package for dev/testing purposes.
...	other variables that called methods can check if they want

Details

This function is strange in that we both want to verify the objects, and return them in some canonical form, so it is normal for the caller to then use the values for x, design, and contrast that are returned from this call, and not the original values for these objects themselves

I know that the validation/checking logic is a bit painful (and repetitive) here. I will (perhaps) clean that up some day.

Value

A list with "normalized" versions of $x, $design, and $contrast for downstream use.

Examples

```
dge.stats <- exampleDgeResult()
ranks <- setNames(dge.stats$t, dge.stats$feature_id)
gdb <- exampleGeneSetDb()
ok <- validateInputs(ranks, gdb, methods = c("cameraPR", "fgsea"))
# need full expressionset & design for romer
null <- failWith(NULL, validateInputs(ranks, gdb, methods = "romer"))
```

---

volcanoPlot *Create an interactive volcano plot*

Description

Convenience function to create volcano plots from results generated within this package. This is mostly used by {sparrow.shiny}.

Usage

```
volcanoPlot(
  x,
  stats = "dge",
  xaxis = "logFC",
  yaxis = "pval",
  idx,
  xtfrm = base::identity,
  ytfrm = function(vals) -log10(vals),
  xlab = xaxis,
  ylab = sprintf("-log10(%s)", yaxis),
  highlight = NULL,
  horiz_line = c(padj = 0.1),
  xhex = NULL,
  yhex = NULL,
  width = NULL,
  height = NULL,
  shiny_source = "mgvolcano",
  ggtheme = ggplot2::theme_bw(),
  ...
)
```

Arguments

x	A SparrowResult object, or a data.frame
stats	One of "dge" or resultNames(x)
xaxis, yaxis	the column of the the provided (or extracted) data.frame to use for the xaxis and yaxis of the volcano
idx	The column of the data.frame to use as the identifier for the element in the row. You probably don't want to mess with this
xtfrm	A function that transforms the xaxis column to an appropriate scale for the x-axis. This is the identity function by default, because most often the logFC is plotted as is.
ytfrm	A function that transforms the yaxis column to an appropriate scale for the y-axis. This is the -log10(yval) function by default, because this is how we most often plot the y-axis.
xlab, ylab	x and y axis labels
highlight	A vector of featureIds to highlight, or a GeneSetDb that we can extract the featureIds from for this purpose.
horiz_line	A (optionally named) number vecor (length 1) that indicates where a line should be drawn across the volcano plot. This is usually done to signify statistical significance. When the number is "named", this indicates that you want to find an approximation of the values plotted on y based on some transformation of the values that is the named column of x (like "padj"). The default value c(padj = 0.10) indicates you want to draw a line at approximately where the adjust pvalue of 0.10 is on the y-axis, which is the *nominal* pvalues.

xhex	The raw .xv (not xtfrm(.xv)) value that acts as a threshold such that values less than this will be hexbinned.
yhex	the .yvt value threshold. Vaues less than this will be hexbinned.
width, height	the width and height of the output plotly plot
shiny_source	the name of this element that is used in shiny callbacks. Defaults to "mggenes".
ggtheme	a ggplot theme, like the thing returned from ggplot2::theme_bw(), for instance.
...	pass through arguments (not used)

Value

a ploty plot object

Examples

```
mg <- exampleSparrowResult()
volcanoPlot(mg)
volcanoPlot(mg, xhex=1, yhex=0.05)
```

---

volcanoStatsTable *Extracts x and y axis values from objects to create input for volcano plot*

Description

You can, in theory, create a volcano plot from a number of different parts of a SparrowResult() object. Most often you want to create a volcano plot from the differential expressino results, but you could imagine building a volcan plot where each point is a geneset. In this case, you would extract the pvalues from the method you like in the SparrowResult() object using the stats parameter.

Usage

```
volcanoStatsTable(
  x,
  stats = "dge",
  xaxis = "logFC",
  yaxis = "pval",
  idx = "idx",
  xtfrm = identity,
  ytfrm = function(vals) -log10(vals)
)
```

Arguments

x	A SparrowResult object, or a data.frame
stats	One of "dge" or resultNames(x)
xaxis, yaxis	the column of the the provided (or extracted) data.frame to use for the xaxis and yaxis of the volcano
idx	The column of the data.frame to use as the identifier for the element in the row. You probably don't want to mess with this
xtfrm	A function that transforms the xaxis column to an appropriate scale for the x-axis. This is the identity function by default, because most often the logFC is plotted as is.
ytfrm	A function that transforms the yaxis column to an appropriate scale for the y-axis. This is the -log10(yval) function by default, because this is how we most often plot the y-axis.

Details

Like the volcanoPlot() function, this is mostly used by the *sparrow.shiny* package.

Value

a data.frame with .xv, .xy, .xvt and .xvy columns that represent the xvalues, yvalues, transformed xvalues, and transformed yvalues, respectively

Examples

```
mg <- exampleSparrowResult()
v.dge <- volcanoStatsTable(mg)
v.camera <- volcanoStatsTable(mg, 'camera')
```

---

zScore *Calculate single sample geneset score by average z-score method*

Description

Calculate single sample geneset score by average z-score method

Usage

```
zScore(x, summary = c("mean", "sqrt"), trim = 0, ...)
```

Arguments

x	gene x sample matrix with rows already subsetted to the ones you care about.
summary	sqrt or mean
trim	calculate trimmed mean?
...	pass through arguments

Value

A list of stats related to the zscore. You care mostly about `$score`.

Examples

```
vm <- exampleExpressionSet(do.voom=TRUE)
gdb <- conform(exampleGeneSetDb(), vm)
features <- featureIds(gdb, 'c2', 'BURTON_ADIPOGENESIS_PEAK_AT_2HR',
                       value='x.idx')
zscores <- zScore(vm[features,])

## Use scoreSingleSamples to facilitate scoring of all gene sets
scores.all <- scoreSingleSamples(gdb, vm, 'zscore', summary = "mean")
s2 <- with(subset(scores.all, name == 'BURTON_ADIPOGENESIS_PEAK_AT_2HR'),
           setNames(score, sample_id))
all.equal(s2, zscores$score)
```

`[,GeneSetDb,ANY,ANY,ANY-method`

Subset whole genesets from a GeneSetDb

Description

Subset whole genesets from a GeneSetDb

Usage

```
## S4 method for signature 'GeneSetDb,ANY,ANY,ANY'
x[i, j, ..., drop = FALSE]
```

Arguments

`x`	GeneSetDb
`i`	a logical vector as long as `nrow(geneSets(x))` indicating which geneSets to keep
`j`	ignored
`...`	pass through arguments
`drop`	ignored

Value

GeneSetDb `x` with a subset of the genesets it came in with.k

Index