Package 'geva'

January 20, 2021

Type Package

Title Gene Expression Variation Analysis

Description Statistic methods to evaluate variation between multiple biological conditions.

Version 0.1.0

Author Nunes I.J.G. <nunesijg@gmail.com>, David M. Z., Feltes B. C., and Dorn M.

Maintainer Nunes I.J.G. <nunesijg@gmail.com>

License LGPL (>=3)

Encoding UTF-8

Depends R (>= 3.5.0)

Imports dbscan, fastcluster, matrixStats, sp, SearchTrees

Suggests knitr, rmarkdown, roxygen2, limma, topGO

R topics documented:

`geva.cluster` *GEVA Cluster Analysis*

Description

Performs a cluster analysis from summarized data.

Usage

```
geva.cluster(
  sv,
  cluster.method = options.cluster.method,
  cl.score.method = options.cl.score.method,
  resolution = 0.3,
  distance.method = options.distance,
  ...,
  grouped.return = FALSE
)

options.cluster.method
# c("hierarchical", "density", "quantiles")

options.cl.score.method
# c("auto", "hclust.height", "density", "centroid")

options.distance
# c("euclidean", "manhattan")
```

Arguments

`sv`	a `numeric` `SVTable` object (usually `GEVASummary`)
`cluster.method`	`character`, one of the main grouping methods (see 'Details')
`cl.score.method`	`character`, method used to calculate the cluster scores for each point. Ignored if `cluster.method` is `quantiles`
`resolution`	`numeric` (`0` to `1`), used as a "zoom" parameter for cluster detection. A zero value returns the minimum number of clusters that can detected by the `cluster.method`, while `1` returns the maximum amount of clusters. Ignored if `cluster.method` is `quantiles`
`distance.method`	`character`, two-point distance calculation method. Options are `"eucludian"` or `"manhattan"` distances
`...`	further arguments passed to `geva.dcluster()`, `geva.hcluster()`, or `geva.quantiles()`. In addition, the following arguments are accepted: • `eps` : `numeric`, defines the *epsilon* coefficient for density clustering (see 'Details') • `mink.p` : `numeric`, parameter for the Minkowsky metric used in hierarchial clustering. Used as the `p` argument for `fastcluster::hclust.vector()` • `verbose` : `logical`, whether to print the current progress (default is `TRUE`)

grouped.return logical, whether to concatenate the clustered and summarized data into a single object

Details

The cluster.method determines which grouping subroutine is used to classify the summarized data points based on distance and partitioning. Each option has their equivalent functions that can be called directly: "density" uses geva.dcluster(); "hierarchical" uses geva.hcluster(); and "quantiles" calls geva.quantiles(). However, this wrapper function can also be used to join GEVASummary and GEVAGroupSet objects into a single GEVAGroupedSummary object by setting grouped.return to TRUE.

The cl.score.method argument defines how scores are calculated for each SV point (row in sv) that was assigned to a cluster, (*i.e.*, excluding non-clustered points). If specified as "auto", the parameter will be selected based on the cluster.method: "density" (rate of neighbor points) for the density method; and "hclust.height" (local hierarchy height) for the hierarchical method. The "centroid" method calculates the scores based on the proportional distance between each point to its cluster's centroid. Note that the cl.score.method argument is ignored if cluster.method is "quantiles", since quantile scores are always based on their local centroid distances.

The resolution value is a more accessible way to define the cluster separation threshold used in density and hierarchical clustering methods. Density clusters uses an *epsilon* value that represents the minimum distance of separation, whereas hierarchical clusters are defined by cutting the hierarchy tree wherever there is a minimum distance between two hierarchies. In this sense, resolution translates a value between 0 and 1 to propotional value for *epsilon* or hierarchical height (depending on the cluster.method) that would result in the least number of possible clusters for 0 and the highest number for 1. Nevertheless, if *epsilon* is specified as eps in the optinal arguments, its value is used and resolution is ignored.

Value

This function produces a GEVAGroupSet-derived object, particularly a GEVACluster for the "hierarchical" and "density" cluster methods or a GEVAQuantiles for the "quantiles" method.

However, if grouped.return is TRUE and sv is a GEVASummary object, the produced GEVAGroupSet data will be concatenated to the input and returned as a GEVAGroupedSummary

See Also

Other geva.cluster: geva.dcluster(), geva.hcluster(), geva.quantiles()

Examples

```
## Cluster analysis from a randomly generated input

# Preparing the data
ginput <- geva.ideal.example()      # Generates a random input example
gsummary <- geva.summarize(ginput)  # Summarizes with the default parameters

# Hierarchical clustering
gclust <- geva.cluster(gsummary, cluster.method="hierarchical")
plot(gclust)

# Density clustering
gclust <- geva.cluster(gsummary, cluster.method="density")
plot(gclust)
```

```
# Density clustering with slightly more resolution
gclust <- geva.cluster(gsummary,
                       cluster.method="density",
                       resolution=0.35)
plot(gclust)
```

`geva.dcluster` *GEVA Density Clustering*

Description

Performs a density cluster analysis from summarized data.

Usage

```
geva.dcluster(
  sv,
  resolution = 0.3,
  dcluster.method = options.dcluster.method,
  cl.score.method = options.cl.score.method,
  minpts = 2,
  ...,
  eps = NA_real_,
  include.raw.results = FALSE
)

options.dcluster.method
# c("dbscan", "optics")
```

Arguments

`sv`	a `numeric` `SVTable` object (usually `GEVASummary`)
`resolution`	`numeric` (`0` to `1`), used as a "zoom" parameter for cluster detection. A zero value returns the minimum number of clusters that can detected, while `1` returns the maximum amount of detectable clusters. Ignored if `eps` is specified
`dcluster.method`	`character`, density-based method for cluster separation
`cl.score.method`	`character`, method used to calculate the cluster scores for each point. If `"auto"`, the `"density"` method is selected
`minpts`	`integer`, minimum number of points required to form a cluster
`...`	additional arguments. Accepts `verbose` (`logical`, default is `TRUE`) to enable or disable printing the current progress
`eps`	`numeric`, maximum neighborhood distance between points to be clustered
`include.raw.results`	`logical`, whether to attach intermediate results to the returned object

Details

This function performs a density cluster analysis with the aid of implemented methods from the dbscan::dbscan package. The available methods for the dcluster.method arguments are "dbscan" and "options", which internally call dbscan::dbscan() and dbscan::optics(), respectively.

The resolution value is an accessible way to define the cluster separation threshold used in density clustering. The *DBSCAN* algorithm uses an *epsilon* value that represents the minimum distance of separation, and resolution translates a value between 0 and 1 to a propotional value within the acceptable range of *epsilon* values. This allows defining the rate of clusters from 0 to 1, which results in the least number of possible clusters for 0 and the highest number for 1. Nevertheless, if *epsilon* is specified as eps in the optinal arguments, its value is used and resolution is ignored.

The cl.score.method argument defines how scores are calculated for each SV point (row in sv) that was assigned to a cluster, (*i.e.*, excluding non-clustered points). If specified as "auto", the parameter will be selected based on the rate of neighbor points ("density").

If include.raw.results is TRUE, some aditional data will be attached to the info slot of the returned GEVACluster objects, including the *kNN* tree generated during the intermediate steps.

Value

A GEVACluster object

Note

In density clustering, only the most dense points are clustered. For the unclustered points, the grouping value is set to NA.

See Also

Other geva.cluster: geva.cluster(), geva.hcluster(), geva.quantiles()

Examples

```
## Density clustering from a randomly generated input

# Preparing the data
ginput <- geva.ideal.example()      # Generates a random input example
gsummary <- geva.summarize(ginput)  # Summarizes with the default parameters

# Density clustering
gclust <- geva.dcluster(gsummary)
plot(gclust)

# Density clustering with slightly more resolution
gclust <- geva.dcluster(gsummary, resolution=0.35)
plot(gclust)
```

`geva.finalize` *Concatenating GEVA calculations into the final results*

Description

Merges the obtained information (Summarization, Clustering, and Quantiles), then applies the final steps to produce the classification results for the SV points (genes).

Usage

```
geva.finalize(
  gsummary,
  ...,
  p.value = 0.05,
  p.val.adjust = options.factoring.p.adjust,
  constraint.factors = TRUE
)

options.factoring.p.adjust
# c("partial.quantiles", "holm", "hochberg", "hommel",
#   "bonferroni", "BH", "BY", "fdr", "none")
```

Arguments

`gsummary`	a `GEVASummary` object
`...`	Intermediate results produced from the `gsummary` object, such as clusters (`GEVACluster`), quantiles (`GEVAQuantiles`), or any other object inherited from `GEVAGroupSet`
`p.value`	`numeric` (0 to 1), p-value cutoff used in the ANOVA procedures (factor analysis only)
`p.val.adjust`	`character`, p-value correction method (factor analysis only). Possible values are: "partial.quantiles", "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none"
`constraint.factors`	`logical`. If `TRUE`, the S values are restricted to the range within the quantile centroids (factor analysis only)

Details

In this procedure, the SV points (*i.e.*, each row in the `GEVASummary` object) are classified according to the detected quantiles (see geva.quantiles), whose results can be adjusted using other grouping analysis results such as clusters (see geva.cluster). To achieve the best statistical accuracy, both `GEVAQuantiles` and `GEVACluster` objects must be given in the `...` as optional arguments. If a `GEVAQuantiles` argument is not present, it is automatically calculated using the default parameters.

If multiple factors are present in the `GEVASummary` object (retrieved by `factors(gsummary)`), a factor analysis is also performed, giving two additional possible classifications (*factor-dependent* and *factor-specific*) besides the default ones (*similar*, *basal*, and *sparse*).

In factor analysis, an ANOVA is applied for each gene using Fisher's and Levene's tests to distinguish genes whose *logFC* (differential expression) variation is dependent or specific to the analyzed factors based on the p-value cutoff. The `p.val.adjust` argument defines how these p-values will

be adjusted: by quantile separation between each factor (`"partial.quantiles"` method); or by one of the default methods listed in stats::p.adjust.methods.

The `constraint.factors` argument determines if the `S` values (summarized *logFC*) will be limited to the range between the quantile centroids during factor analysis. For example, if the quantile centroids were -0.90, 0.00, and 0.90 in the S axis, values such as -1.53 and 2.96 would be converted to -0.90 and 0.90, respectively. This constraint is particularly applied to avoid significative observations from ANOVA based on multiple degrees of differential expression.
In another example to illustrate the constraint of factors, given two sets of values: A = (-1.00, -1,10, 0.00, 0.20, 1.00, 1.15), and B = (0.00, 0.12, 1.11, 1.00, 1.95, 2.00), with the centroids located in C = (-0.90, 0.00, 0.90), and the factors F = (Cond1, Cond1, Cond2, Cond2, Cond3, Cond3). If `constraint.factors` is `FALSE`, both A and B are considered as significantly separated factors, whereas if `TRUE`, only A will present a significant separation, since in B the values 1.11, 1.00, 1.95, and 2.00 are converted to 0.90. In qualitative terms, if `constraint.factors` is `TRUE`, all values above 0.90 are considered the same over-expressed values, ensuring that they will fit in the same degree of differential expression. Hence, in this example using the constrained values, B would not represent a significant separation between the factors Cond1, Cond2, and Cond3.

Value

A `GEVAResults` object, containing the entire set of results. The relevant genes can be retrieved using `top.genes()`

Note

To perform factor analysis, the following observations must be considered:

- The factors must be defined in the provided data. They can be retrieved using the `factors` accessor. If factors are not present or are entirelly composed by `NA`, they can be assigned through `factors<-` by providing a `factor` or `character` vector of the same length of the input columns;
- Each factor must include two or more values, since the factor analysis is based on ANOVA and at least two values are needed to variance calculation;
- Columns whose factor value is `NA` are not considered.

See Also

p.adjust.methods

Examples

```
## Finalizing example using a random generated input
ginput <- geva.ideal.example()      # Generates a random input (for testing purposes only)
gsummary <- geva.summarize(ginput)   # Summarizes the input
gquant <- geva.quantiles(gsummary)   # Calculates the quantiles
gclust <- geva.cluster(gsummary)     # Calculates the clusters
gresults <- geva.finalize(gsummary, gquant, gclust)  # Finishes the results

head(top.genes(gresults))            # Prints the final results
plot(gresults)                       # Plots the final SV-plot
```

`geva.hcluster` *GEVA Hierarchical Clustering*

Description

Performs a hierarchical cluster analysis from summarized data.

Usage

```
geva.hcluster(
  sv,
  resolution = 0.3,
  hc.method = options.hc.method,
  hc.metric = options.hc.metric,
  cl.score.method = options.cl.score.method,
  ...,
  include.raw.results = FALSE
)

options.hc.metric
# c("euclidean", "maximum", "manhattan", "canberra",
#   "binary", "minkowski")

options.hc.method
# c("centroid", "median", "ward", "single")
```

Arguments

`sv`	a `numeric` `SVTable` object (usually `GEVASummary`)
`resolution`	`numeric` (`0` to `1`), used as a "zoom" parameter for cluster detection. A zero value returns the minimum number of clusters that can detected, while `1` returns the maximum amount of detectable clusters
`hc.method`	`character`, the agglomeration method to be used. Used as the `method` argument for `fastcluster::hclust.vector()`
`hc.metric`	`character`, the distance measure to be used. Used as the `metric` argument for `fastcluster::hclust.vector()`
`cl.score.method`	`character`, method used to calculate the cluster scores for each point. If `"auto"`, the `"hclust.height"` method is selected
`...`	additional arguments: • `mink.p` : `numeric`, parameter for the Minkowsky metric. Used as the `p` argument for `fastcluster::hclust.vector()` • `verbose` : `logical`, whether to print the current progress (default is `TRUE`)
`include.raw.results`	`logical`, whether to attach intermediate results to the returned object

Details

This function performs a hierarchical cluster analysis with the aid of implemented methods from the fastcluster::fastcluster package, particularly the fastcluster::hclust.vector() function. The available methods for the hc.method and hc.metric are described in the function's documentation page (see fastcluster::hclust.vector()).

The resolution value is an accessible way to define the cluster separation threshold used in hierarchical clustering. The algorithm produces a dendrogram-like hierarchy in which each level/node is separated by a distance (sometimes called "height") to the next level/node, and the resolution translates a value between 0 and 1 to a propotional value within the total hierarchy height. This allows defining the rate of clusters from 0 to 1, which results in the least number of possible clusters (usually two) for 0, and the highest number (approximately one cluster per point) for 1.

If include.raw.results is TRUE, some aditional data will be attached to the info slot of the returned GEVACluster objects, including the *kNN* tree generated during the intermediate steps.

Value

A GEVACluster object

Note

In hierarchical clustering, all points are clustered. Therefore, setting resolution to 1 will result into one cluster per point, where the cluster analysis may become pointless (no pun intended).

See Also

Other geva.cluster: geva.cluster(), geva.dcluster(), geva.quantiles()

Examples

```
## Hierarchical clustering from a randomly generated input

# Preparing the data
ginput <- geva.ideal.example()      # Generates a random input example
gsummary <- geva.summarize(ginput)  # Summarizes with the default parameters

# Hierarchical clustering
gclust <- geva.hcluster(gsummary)
plot(gclust)

# Hierarchical clustering with slightly more resolution
gclust <- geva.hcluster(gsummary,
                        resolution=0.35)
plot(gclust)
```

---

geva.ideal.example *GEVA "Ideal" Example for Package Testing*

Description

Generates a random example of GEVAInput object that simulates an ideal analysis dataset. Used for testing purposes only.

Usage

```
geva.ideal.example(probecount = 10000, nfactors = 3, colsperfactor = 3)
```

Arguments

probecount	integer, number of probes (*i.e.*, table rows)
nfactors	integer, number of factors (*e.g.*, experimental groups)
colsperfactor	integer, number of columns (*e.g.*, experiments) per factor

Value

A GEVAInput object. The included tables are composed by probecount rows and nfactors * colsperfactor columns

See Also

geva.summarize

Examples

```
## "Ideal" input example
ginput <- geva.ideal.example()     # Generates a random example
gsummary <- geva.summarize(ginput) # Summarizes the generated data
plot(gsummary)                     # Plots the summarized data
```

---

geva.input.correct *GEVA Input Post-processing*

Description

Helper functions used to edit the contents from a GEVAInput.

Usage

```
geva.input.correct(ginput, na.rm = TRUE, inf.rm = TRUE, invalid.col.rm = TRUE)

geva.input.filter(
  ginput,
  p.value.cutoff = 0.05,
  by.any = FALSE,
  na.val = 0,
  ...
)

geva.input.rename.rows(
  ginput,
  attr.column,
  dupl.rm.method = c("least.p.vals", "order")
)
```

Arguments

ginput	A GEVAInput object
na.rm	logical; if TRUE, removes all rows containing NA
inf.rm	logical; if TRUE, removes all rows containing infinite values (Inf or -Inf)
invalid.col.rm	logical; if TRUE, searches for any column that is entirely composed by invalid values (according to the other arguments) and removes it before checking the rows
p.value.cutoff	numeric (0 to 1), the p-value cutoff. Rows containing values above this threshold are removed
by.any	logical, set to TRUE to delete the rows with at least one occurrence above the cutoff; or FALSE to delete only those rows in which all values are above the specified threshold
na.val	numeric, the replacement for NA values
...	optional arguments. Accepts verbose (logical, default is TRUE) to enable or disable printing the progress
attr.column	character, target column with the values that will replace the current row names
dupl.rm.method	character, method to remove duplicate names. The possible options are: • "least.p.vals" : Keeps the duplicate that contains the least sum of p-values • "order" : Keeps the first occurrence of the duplicate in the current row order

Details

geva.input.correct corrects the numeric input data (values and weights), removing rows that include invalid values such as NA or infinite.

geva.input.filter attempts to select the most relevant part of the input data, removing rows containing p.values (1 - weights) above a specific threshold.

geva.input.rename.rows replaces the row names with a column from the feature table (see GEVAInput). The column name specified for attr.column must be included in the names(featureTable(ginput)). Any duplicates are removed according to the dupl.rm.method, and the selected duplicates are stored as a new column named "renamed_id" inside the feature table from the returned object.

Value

A modified GEVAInput object

Examples

```
## geva.input.correct example
colexample1 <- runif(1000, -1, 1)        # Random column 1
colexample2 <- runif(1000, -1, 1)        # Random column 2
colexample3 <- runif(1000, -1, 1)        # Random column 3
colexample3[runif(1000, -1, 1) < 0] = NA # Random NA's
ginput = geva.merge.input(col1=colexample1,
                          col2=colexample2,
                          col3=colexample3)
# Before the correction:
```

```
print(nrow(ginput))    # Returns 1000
# Applies the correction (removes rows with NA's)
ginput <- geva.input.correct(ginput)
# After the correction:
print(nrow(ginput))    # Returns less than 1000

## ---
## geva.input.filter example
ginput <- geva.ideal.example(1000)  # Generates a random input
# Before the filter:
print(nrow(ginput))    # Returns 1000
# Applies the filter
ginput <- geva.input.filter(ginput)
# After the filter:
print(nrow(ginput))    # Returns less than 1000

## ---
## geva.input.rename.rows example
ginput <- geva.ideal.example()  # Generates a random input
# Renames to 'Symbol'
ginput <- geva.input.rename.rows(ginput,
                                 attr.column = "Symbol")
print(head(ginput))             # The row names are set now as the gene symbols
```

`geva.merge.input` *GEVA Input Processing and Merge*

Description

Functions to read, load, and concatenate the experimental comparisons from the data input. This is the initial step to proceed with any GEVA analysis.

Usage

```
geva.merge.input(
  ...,
  col.values = "logFC",
  col.pvals = "adj.P.Val",
  col.other = NULL
)

geva.read.tables(
  filenames = NULL,
  dirname = ".",
  col.values = "logFC",
  col.pvals = "adj.P.Val",
  col.other = NULL,
  ...,
  files.pattern = "\\.txt$",
  p.value.cutoff = 0.05,
  read.args = list()
)
```

Arguments

`...`	multiple `matrix` or `data.frame` objects. At least two arguments are required for `geva.merge.input`, but it's optional for `geva.read.tables`. The optional arguments in `geva.read.tables` are also passed to its internal call to `geva.merge.input` and `geva.input.filter`. In addition, the following optional arguments are accepted: • `na.val` : (`numeric`) value between `0` and `1` used as replacement when a p-value column is not present (default is `NA`) • `use.regex` : (`logical`) whether to match the column names using regular expressions (default is `FALSE`) • `verbose` : (`logical`) whether to print the current loading and merge progress (default is `TRUE`)
`col.values`	`character` vector, possible name(s) to match the *logFC* column(s) from each table
`col.pvals`	`character` vector, possible name(s) to match the p-value column(s) from each table
`col.other`	`character` vector, name(s) to match additional columns (*e.g.*, gene symbols). Ignored if `NULL`
`filenames`	`character` vector with two or more file paths
`dirname`	single `character`, base directory containing the input files. Ignored if `filenames` is specified
`files.pattern`	single `character`, pattern used to filter the files inside `dirname`. Ignored if `filenames` is specified
`p.value.cutoff`	`numeric` (`0` to `1`), initial p-value threshold. Rows entirely composed by p-values above this cutoff (*i.e.*, no significant *logFC*) are removed after the final merge. Ignored if `NA` or `NULL`
`read.args`	`list` of additional arguments passed to utils::read.table

Details

The `geva.merge.input` function takes multiple tables as arguments (*e.g.*, `matrix` or `data.frame` objects), extracts the *logFC* columns from each table and merges them into a single `GEVAInput` dataset.

The column names are specified in the `col.values` and `col.pvals` arguments (`character`) and must correctly match the column names for *logFC* and p-value columns, respectively, in the inputs to be extracted. Multiple values for column names can also be specified as valid name possibilities if they differ among the tables.

The function `geva.merge.input` reads multiple tab-delimited text files containing, extracts the *logFC* columns from each table and merges into a single `GEVAInput` dataset.

Value

A `GEVAInput` object

Note

The inclusion of p-value columns is not technically required, but strongly recommended as they improve the statistical accuracy in the summarization steps. If the p-value (or adjusted p-value) columns are present, their values are converted to weights by applying `1 -pvalue` for each `pvalue`

element, otherwise an optional na.val optional argument can specified as replacement to the absent values (default is NA). Weights are used to accomodate the central *logFC* values towards the most significant observations and penalize potential statistical innacuracies.

Examples

```
### EXAMPLE 1
## geva.merge.input example with three randomly generated tables
## (For demonstration purposes only)

# Number of rows
n <- 10000

# Random row (probe) names
probnms <- sprintf("PROBE_%s", 1:n)

# Random gene names (optional)
genenms <- paste0(sprintf("GENE_%s", 1:n), LETTERS[1:n %% (length(LETTERS)+1)])

# Random table 1
dt1 <- data.frame(row.names=probnms,
                  logfc=(rnorm(n, 0, sd=2) * rnorm(n, 0, sd=0.5)),
                  pvalues = runif(n, max=0.08),
                  genesymbol = genenms)
# Random table 2
dt2 <- data.frame(row.names=probnms,
                  logfc=(rnorm(n, 0, sd=2) * rnorm(n, 0, sd=0.5)),
                  pvalues = runif(n, max=0.08),
                  genesymbol = genenms)
# Random table 3
dt3 <- data.frame(row.names=probnms,
                  logfc=(rnorm(n, 0, sd=2) * rnorm(n, 0, sd=0.5)),
                  pvalues = runif(n, max=0.08),
                  genesymbol = genenms)

# Merges the three tables
ginput <- geva.merge.input(exp1=dt1, exp2=dt2, exp3=dt3,
                           col.values="logfc",
                           col.pvals="pvalues",
                           col.other="genesymbol")

# Prints the first rows from the merged table
print(head(ginput))               # values
print(head(inputweights(ginput))) # weights

# ---
## Not run:

### EXAMPLE 2
## geva.read.tables example with three tab-delimited files

# Table file examples. Each one has 3 columns: "logfc", "pvalues", and "genesymbol"
# Replace it with your tab-delimited files (e.g. exported from limma's topTable)
fnames <- c("dt1.txt", "dt2.txt", "dt3.txt")

ginput <- geva.read.tables(fnames,
                           col.values="logfc",
```

```
                            col.pvals="pvalues",
                            col.other="genesymbol")

# Prints the first rows from the merged table
print(head(ginput))               # values
print(head(inputweights(ginput))) # weights


# ---

### EXAMPLE 3
## geva.read.tables example with tab-delimited files in a directory

# Directory name (replace it with a directory containing the table files)
dirnm <- "C:/User/robertplant123/Documents/R/gevaexamples"

# In this example, table files contain 3 columns: "logfc", "pvalues", and "genesymbol"
# Reads all txt files in the directory
ginput <- geva.read.tables(dirname=dirnm,
                            col.values="logfc",
                            col.pvals="pvalues",
                            col.other="genesymbol")

# (Optional step)
# Let's assume that all table file names start with "dt" and ends with the ".txt" extension,
# such as dt1.txt, dt2.txt and so on...
fname_pattern <- c("^dt.+?\\.txt$")  # Defines a RegEx pattern to find the files
# Loads only files that match the file name pattern
ginput <- geva.read.tables(dirname=dirnm,
                            files.pattern=fname_pattern,
                            col.values="logfc",
                            col.pvals="pvalues",
                            col.other="genesymbol")

# Prints the first rows from the merged table
print(head(ginput))               # values
print(head(inputweights(ginput))) # weights

## End(Not run)
```

`geva.quantiles` *GEVA Quantiles Detection Calculates the quantiles of a* `SVTable`

Description

GEVA Quantiles Detection

Calculates the quantiles of a `SVTable`

Returns a vector with the supported methods of quantiles separation

Usage

```
geva.quantiles(
  sv,
```

```
  quantile.method = options.quantiles,
  initial.thresholds = c(S = NA_real_, V = NA_real_),
  nq.s = 3L,
  nq.v = 2L,
  comb.score.fn = prod,
  ...
)

options.quantiles
# c("range.slice", "proportional", "density", "k.max.sd",
#   "custom")
```

Arguments

`sv`	a `SVTable` object (usually `GEVASummary`)
`quantile.method`	`character`, method to detect the initial quantile thresholds. Ignored if `initial.thresholds` is specified with no `NA` elements
`initial.thresholds`	named `numeric` vector with the threshold that delimits the initial quantile
`nq.s`	`integer`, number of quantiles in S-axis (experimental, see 'Note')
`nq.v`	`integer`, number of quantiles in V-axis (experimental, see 'Note')
`comb.score.fn`	`function` applied to merge `S` and `V` score columns into a single column. The function must require only one argument of numeric vector type and return a single `numeric` value. Examples include `prod` or `mean`
`...`	additional arguments include: • `qslice` : `numeric` (`0` to `1`), the axis fraction used by `"range.slice"` and `"density"` methods (see 'Details'). Default is `0.25` • `k` : `integer`, neighbor points used by `"density"` and `"k.max.sd"` methods (see 'Details'). Default is `16` • `verbose` : `logical`, whether to print the current progress. Default is `TRUE`

Details

The `quantile.method` defines how the initial quantile (usually the one at the bottom center) is calculated. Each method has a specific way to estimate the first spatial delimiter, as described below:

`"range.slice"` **(default)** Separation is set at the nearest point to a fraction of the spatial range. This fraction can be specified by the `qslice` optional argument (`numeric`, default is 0.25, or 25%);

`"density"` Separation is set at the point with the most proportional density by *k* neighbor points to its current spatial fraction. This method uses the optional arguments `qslice` (`numeric`, default is 0.25, or 25%) for the desired spatial fraction, and `k` (`numeric`, default is `16`) for the number of neighbor points;

`"k.max.sd"` Separation is set at the point with the greatest standard deviation of distance to its *k* neighbor points. The number of neighbor points can be specified by the `k` optional argument (`numeric`, default is `16`);

`"proportional"` Separation is set at the exact axis division so that all quantiles have the size;

`"custom"` Uses the values specified in the `initial.thresholds` argument.

A custom initial separation point can be specified in the `initial.thresholds` as a `numeric` vector of two elements, where the first element refers to `S` axis and the second, to `V` axis. If one of the elements is `NA`, the initial quantile is calculated for that axis only. If both values are not `NA`, the quantile separation method is ignored and automatically set to `"custom"`.

The `nq.s` and `nq.v` arguments determine the number of quantiles for the `S` and `V` axes, respectively. These parameters can be used to increase the number of possible partitions in the SV space, but their applicability is currently being tested (see 'Note').

The `comb.score.fn` is a function applied to the partial scores for each SV point to combine them into a single value. The result value is defined as the "quantile score" for a SV point. The function is applied iteratively to two-element `numeric` vectors.

Note

Customizing the number of quantiles by `nq.s` and `nq.v` is a **experimental feature** and the remaining analysis steps are mostly based on the default parameters for these arguments. Tests are being conducted to determine this feature's applicability for the next releases.

See Also

geva.cluster

Other geva.cluster: `geva.cluster()`, `geva.dcluster()`, `geva.hcluster()`

Examples

```
## Quantile detection from a randomly generated input

# Preparing the data
ginput <- geva.ideal.example()      # Generates a random input example
gsummary <- geva.summarize(ginput)  # Summarizes with the default parameters

# Default usage
gquants <- geva.quantiles(gsummary) # Detects the quantiles
plot(gquants)                       # Plots the quantiles

# Custom initial delimiters
gquants <- geva.quantiles(gsummary,
                          initial.thresholds = c(S=1.00, V=0.5))
plot(gquants)                       # Plots the quantiles

# Quantile detection using densities
gquants <- geva.quantiles(gsummary, quantile.method = 'density')
plot(gquants)                       # Plots the quantiles
```

---

geva.quick *All-In-One Function for GEVA Intermediate Procedures*

Description

Given a `GEVAInput` object, applies the `geva.summarize()`, `geva.quantiles`, `geva.cluster`, and `geva.finalize` in a single call. Optional arguments are passed to the internal calls of these functions.

Usage

```
geva.quick(gobject, ...)
```

Arguments

gobject	A GEVAInput, or any object that returns a GEVAInput upon calling inputdata(gobject) (*e.g.*, GEVASummary or GEVAResults).
...	Optional arguments passed to geva.summarize(), geva.quantiles(), geva.cluster(), and geva.finalize()

Details

This function performs the summarization, quantile detecetion, and clustering of an input data, then merges the results together and, if applicable, performs a factor analysis. If the gobject is not a GEVAInput, it must provide a valid GEVAInput object when called by inputdata(gobject). Moreover, all parameters used in previous analysis will be taken into account. For instance, if gobject is a GEVASummary obtained by using variation.method='mad', the internal call to geva.summarize in this function will use variation.method='mad' as well, unless if another parameter for variation.method is specified in the ... arguments.

Therefore, this function can be useful not only as a shortcut to analyze GEVAInput but also for parameter testing when applied to a GEVAResults object, since the previous parameters are reused, while the specified parameters are overriden.

Examples

```
## Basic usage using a random generated input
ginput <- geva.ideal.example()   # Generates a random input example
gresults <- geva.quick(ginput)   # Performs the entire analysis (default parameters)

print(head(top.genes(gresults))) # Prints the results
plot(gresults)                        # Plots the final SV-plot


## Example with non-default parameters
ginput <- geva.ideal.example()   # Generates a random input example
gresults <- geva.quick(ginput,
                       summary.method="median",
                       variation.method="mad",
                       quantiles.method="density",
                       cluster.method="density",
                       resolution=0.32)

print(head(top.genes(gresults))) # Prints the results
plot(gresults)                   # Plots the final SV-plot
```

---

geva.summarize *Summarizes the GEVAInput*

Description

Performs the summarization step by calculating the central points and variation estimates of *logFC* values from the input data.

Usage

```
geva.summarize(
  ginput,
  summary.method = options.summary,
  variation.method = options.variation,
  ...
)

options.summary
# c("mean", "median")

options.variation
# c("sd", "var", "mad")
```

Arguments

ginput	a GEVAInput object
summary.method	single character, method used to calculate the central (summarized) *logFC* values
variation.method	single character, method used to calculate the distribution degree (variation) of the *logFC* values
...	additional arguments. Accepts verbose (logical, default is TRUE) to enable or disable printing the current progress

Details

The options.summary refer to the available operations to calculate central *logFC* values (mean or median), whereas options.variation presents three functions to calculate *logFC* variation (sd: Standard Deviation; var: Variance; and mad: Median Absolute Deviation). Moreover, all those operations include a weighted counterpart applied using the weights table from the GEVAInput object.

Value

A GEVASummary object

See Also

base::mean(), stats::median()

stats::var(), stats::sd(), stats::mad()

Examples

```
## Summarization of a randomly generated input
ginput <- geva.ideal.example()     # Generates a random input example
gsummary <- geva.summarize(ginput) # Summarizes with the default parameters
plot(gsummary)                     # Plots the summarized data
```

GEVACluster-class *GEVA Clustering Results*

Description

The GEVACluster class represents the classification results from a cluster analysis. For each probe/gene, there is a assigned cluster among the *g* defined clusters.

This class inherits from GEVAGroupSet.

Slots

grouping factor (*m* elements, *g* levels), cluster assignment for each gene/probe
(Inherited from GEVAGroupSet)

scores numeric vector (*m* elements) comprising a score value for each cluster assignment
(Inherited from GEVAGroupSet)

ftable data.frame (*m* lines) with additional cluster assignment features
(Inherited from GEVAGroupSet)

centroids numeric SVTable (*g* lines) with the S and V centroid coordinates for each cluster
(Inherited from GEVAGroupSet)

offsets numeric SVTable (*m* lines) with the S and V coordinate offsets each gene/probe from its cluster centroid
(Inherited from GEVAGroupSet)

info list of supplementary information
(Inherited from GEVAGroupSet)

cluster.method character, method used in the cluster analysis (see geva.cluster)

Methods

(See also the inherited methods from GEVAGroupSet)

Dimension accessors

lines(x, ...) Draws convex hulls around the clustered points

Slot accessors

cluster.method(object) Gets the character from the cluster.method slot

GEVAGroupedSummary-class
GEVA Grouped Summary-Variation Table

Description

The GEVAGroupedSummary class inherits the GEVASummary class and includes group analysis data (*e.g.*, clustering and quantile detection).

Slots

`sv` `numeric` `matrix` composed by two columns: `S` (summary) and `V` (variation)
(Inherited from SVTable)

`inputdata` GEVAInput-class with the data input
(Inherited from GEVASummary)

`sv.method` Names of the statistical methods used to summarize data
(Inherited from GEVASummary)

`info` list with additional information
(Inherited from GEVASummary)

`groupsetlist` TypedList of GEVAGroupSet objects

Methods

`cluster.method(object)` Gets a `character` vector listing the `cluster.method` from each group set

Dimension accessors

`lines(x, ...)` Draws delimiters within quantiles and convex hulls around the clustered points

Properties

`analysis.params(gobject)` Returns a `list` of analysis parameters passed to `geva.cluster` to obtain this object

Slot accessors

`groupsets(object)` Gets the `TypedList` from the `groupsetlist` slot

`quantiles(object)` Gets the GEVAQuantiles, or `NULL` if not present

GEVAGroupSet-class *GEVA Grouping Results*

Description

The GEVAGroupSet class represents the classification of summarized values from a SVTable, where each gene/probe has one assigned group among g defined groups. This is an abstract class. Inherits the GEVACluster and GEVAQuantiles classes.

Slots

`grouping factor` (*m* elements, *g* levels) used to group the genes/probes

`scores numeric` vector (*m* elements) with the assigned grouping scores for each gene/probe

`ftable data.frame` (*m* lines) with additional grouping features

`centroids numeric SVTable` (*g* lines) with the S and V centroid coordinates for each group

`offsets numeric SVTable` (*m* lines) with the S and V coordinate offsets each gene/probe from its group centroid

`info list` of additional information

Methods

`sv(object)` Returns the numeric matrix in the `SVTable` from `sv.data(object)`

Plotting

`color.values(x, point.col = NULL, ...)` Gets the colors associated to the grouped data points. If not present, generates random group colors.
If `point.col` is a single `character` or an vector of the same length of data points, adjusts the color values to web RGBA

`points(x, which, ..., classif)` Draws the grouped points

Properties

`analysis.params(gobject)` Returns a `list` of analysis parameters passed to `geva.cluster` to obtain this object

Slot accessors

`centroids(object)` Gets the `SVTable` from the `centroids` slot

`featureTable(object)` Gets the `data.frame` from the `ftable` slot

`groups(object)` Gets the `value` from the `groups` slot

`infolist(object, field, ...)` Gets the `list` from the `info` slot.
If `field` is a `character`, returns the element with the matching name (`infolist(object)$<field name>`)

`infolist(object) <- value` Sets a value to the `info` slot

`offsets(object)` Gets the `SVTable` from the `offsets` slot

`scores(object, group)` Gets the `value` from the `If` slot. `scores` is a group name, returns only the scores from this group

`sv.data(object)` Returns a `SVTable` with the source SV coordinates

GEVAInput-class *GEVA Input Data*

Description

The GEVAInput class contains the initial data for GEVA usage. It stores numeric matrices of *logFC* values from differential expression comparison results. Options for calculations and summarizing are also included.

Slots

`values` numeric matrix (*m*n*) of log-ratio values, usually *logFC*

`weights` numeric matrix (*m*n*) of weighted values. If not defined, all weight values are equal to 1

`factors` `factor` (*n* elements) representing the grouping of the n columns. If not defined, all factors are equal to NA

`ftable` `data.frame` with *m* rows containing attribute columns associated to the features (*e.g.*, probes or genes)

`info` `list` of supplementary information related to the input

Methods

Dimension accessors

`dim(x)` Gets the dimensions defined for both matrices in `values` and `weights` slots

`dimnames(x)` Gets a `list` with the row and column names.
Individual dimension names can also be accessed through `rownames` and `colnames`

`dimnames(x) <- value` Sets the `list` with the row and column names.
Individual dimension names can also be set using `rownames<-` and `colnames<-`

Properties

`analysis.params(gobject)` Returns a `list` of analysis parameters passed to `geva.merge.input` or `geva.read.tables` to obtain this object

Slot accessors

`factors(object)` Gets the `factor` from the `factors` slot

`factors(object) <- value` Sets a value to the `factors` slot

`featureTable(object)` Gets the `data.frame` from the `ftable` slot

`featureTable(object) <- value` Sets a value to the `ftable` slot

`infolist(object, field = NULL, ...)` Gets the `list` from the `info` slot.
If `field` is a `character`, returns the element with the matching name (`infolist(object)$<field name>`)

`infolist(object) <- value` Sets a value to the `info` slot

`inputvalues(object)` Gets the `matrix` from the `values` slot

`inputweights(object, normalized = FALSE)` Gets the `matrix` from the `weights` slot. If `normalized` is `TRUE`, returns the weights matrix in the normalized form

Sub-slot accessors

`inputnames(object)` Gets the input column names (same as `colnames(object)`)

GEVAQuantiles-class *GEVA Quantiles Grouping Results*

Description

The `GEVAQuantiles` class represents the results of a quantile detection analysis. For each probe/gene, there is a assigned quantile among the *g* defined quantiles.

This class inherits from `GEVAGroupSet` and is inherited by `GEVAQuantilesAdjusted`.

Slots

`grouping` `factor` (*m* elements, *g* levels), quantile assignment for each gene/probe
(Inherited from `GEVAGroupSet`)

`scores` `numeric` vector (*m* elements) with the assigned quantile scores for each gene/probe
(Inherited from `GEVAGroupSet`)

`ftable` `data.frame` (*m* lines) with additional quantile assignment features
(Inherited from `GEVAGroupSet`)

`centroids` `numeric` `SVTable` (*g* lines) with the S and V centroid coordinates for each quantile
(Inherited from `GEVAGroupSet`)

`offsets` `numeric` `SVTable` (*m* lines) with the S and V coordinate offsets each gene/probe from its quantile centroid
(Inherited from `GEVAGroupSet`)

`info` `list` of additional information
(Inherited from `GEVAGroupSet`)

`svscores` `numeric` `SVTable` (*m* lines) with individual partial scores for the assigned quantiles

`qareasizes` `numeric` `SVTable` (*g* lines) with the S and V sizes for each quantile

`qindexes` `integer` `SVTable` (*g* lines) representing the position index to each quantile, in terms of summary and variation

`qcount` integer attributes (`SVIntAttribute`) with the defined number of quantiles for the S and V axes

`qcutoff` numeric attributes (`SVNumAttribute`) with the initial quantile cutoff in S and V, starting from the point zero

Methods

(See also the inherited methods from `GEVAGroupSet`)

Dimension accessors

`lines(x, ...)` Draws the quantile delimiter lines

Slot accessors

`qareasizes(object)` Gets the `SVTable` from the `qareasizes` slot

`qcount(object)` Gets the `SVIntAttribute` from the `qcount` slot

`qindexes(object)` Gets the `SVTable` from the `qindexes` slot

`quantiles(object)` Gets the unique quantile names

`quantiles.method(object)` Gets the `character` from the `qmethod` slot

`sv.scores(object)` Gets the `SVTable` from the `svscores` slot

`GEVAQuantilesAdjusted-class`

GEVA Adjusted Quantiles Results

Description

The `GEVAQuantilesAdjusted` class represents the results of a quantile detection analysis with adjusted assignments based on relationships with other `GEVAGroupSet` objects. For each probe/gene, there is a assigned quantile among the *g* defined quantiles.

This class inherits from `GEVAQuantiles`.

Slots

`grouping factor` (*m* elements, *g* levels), quantile assignment for each gene/probe
(Inherited from `GEVAGroupSet`)

`scores numeric` vector (*m* elements) with the assigned quantile scores for each gene/probe
(Inherited from `GEVAGroupSet`)

`ftable data.frame` (*m* lines) with additional quantile assignment data
(Inherited from `GEVAGroupSet`)

`centroids numeric SVTable` (*g* lines) with the S and V centroid coordinates for each quantile
(Inherited from `GEVAGroupSet`)

`offsets numeric SVTable` (*m* lines) with the S and V coordinate offsets each gene/probe from its quantile centroid
(Inherited from `GEVAGroupSet`)

`info list` of additional information
(Inherited from `GEVAGroupSet`)

`svscores numeric SVTable` (*m* lines) with individual partial scores for the assigned quantiles
(Inherited from `GEVAQuantiles`)

`qareasizes numeric SVTable` (*g* lines) with the S and V sizes for each quantile
(Inherited from `GEVAQuantiles`)

`qindexes integer SVTable` (*g* lines) representing the position index to each quantile, in terms of summary and variation
(Inherited from `GEVAQuantiles`)

`qcount` integer attributes (`SVIntAttribute`) with the defined number of quantiles for the S and V axes
(Inherited from `GEVAQuantiles`)

qcutoff numeric attributes (SVNumAttribute) with the initial quantile cutoff in S and V, starting from the point zero
(Inherited from GEVAQuantiles)

grouprels TypedList of named factor elements representing external group relationships to the current quantiles

Methods

(See also the inherited methods from GEVAQuantiles and GEVAGroupSet)

Slot accessors

group.rels(object) Gets the TypedList from the grouprels slot

---

GEVAResults-class *GEVA Results Table*

Description

The GEVAResults class contains the final results from GEVA analyses. It represents the results of multiple statistical approaches from summary/variation data, clustering, quantile detection, and factor analysis (if applicable).

Slots

resultstable data.frame (*m* lines) with classification results for the genes/probes

svdata GEVASummary used as input

quantdata GEVAQuantiles or GEVAQuantilesAdjusted with the final quantile assignments for the summarized data

factoring data.frame (*m* lines) with detailed results for the factor analyses, such as p-values for each factor. If there was no factor analysis, this slot is NULL or empty

classiftable data.frame used as reference for the final classification

info list of supplementary information

Methods

Dimension accessors

dim(x) Gets the dimensions from the results.table slot

dimnames(x) Gets a list with the row and column names from the results.table slot.
Individual dimension names can also be accessed through rownames and colnames

Plotting

points(x, which, ..., classif) Draws the results points.
If which (character vector) is given, plots only the matching genes/probes.
If classif (character vector) is given, plots only points with the matching classification

Properties

`analysis.params(gobject)` Returns a `list` of analysis parameters passed to `geva.finalize` or `geva.quick` to obtain this object

Slot accessors

`infolist(object, field, ...)` Gets the `list` from the `info` slot

`infolist(object, field, ...)` Gets the `list` from the `info` slot

`quantiles(object)` Gets the `GEVAQuantiles` from the `quantdata` slot

`results.table(gres)` Gets the `data.frame` from the `resultstable` slot

`sv.data(object)` Gets the `GEVASummary` from the `svdata` slot

Sub-slot accessors

`featureTable(object)` Returns the features `data.frame` from the internal `GEVAInput`

`head(x, ...)` Returns the first lines of `results.table(x)`

`inputdata(object)` Returns the internal `GEVAInput`

`inputvalues(object)` Returns the values `matrix` from the internal `GEVAInput`

`inputweights(object, normalized)` Returns the weights `matrix` from the internal `GEVAInput`

`levels(x)` Returns the factors used in factor analysis, if present

---

GEVASummary-class *GEVA Summary-Variation Table*

Description

The `GEVASummary` class represents the calculation results for summary and variation from a `GEVAInput`.

This class inherits from `SVTable`.

Slots

`sv` `numeric` `matrix` composed by two columns: `S` (summary) and `V` (variation) (Inherited from `SVTable`)

`inputdata` GEVAInput-class with the data input

`sv.method` Names of the statistical methods used to summarize data

`info` list with additional information

Methods

(See also the inherited methods from SVTable

Properties

`analysis.params(gobject)` Returns a `list` of analysis parameters passed to `geva.summarize` to obtain this object

`get.summary.method(gevasummary)` Gets a `character` for the summarization method name

`get.variation.method(gevasummary)` Gets a `character` for the variation calculation method name

Slot accessors

`infolist(object, field, ...)` Gets the `list` from the `info` slot.
If `recursive` is `TRUE`, appends the contents from the `info` slot in the internal `GEVAInput`

`inputdata(object)` Gets the `GEVAInput` from the `inputdata` slot

Sub-slot accessors

`factors(object)` Gets the `factor` defined in the `factors` slot in the internal `GEVAInput`

`factors(object) <- value` Sets the value to the `factor` slot in the internal `GEVAInput`

`featureTable(object)` Gets the `data.frame` from the `ftable` slot in the internal `GEVAInput`

`inputvalues(object)` Gets the `matrix` from the `values` slot in the internal `GEVAInput`

`inputweights(object, normalized)` Gets the `matrix` from the `weights` slot in the internal `GEVAInput`

`SVAttribute-class` *Summary-Variation Attribute Field*

Description

This S4 class stores two character slots representing attribute fields for summary and variation. The `SVAttribute` class is abstract and must be instantiated as `SVChrAttribute` (for `character`), `SVNumAttribute` (for `numeric`), or `SVIntAttribute` (for `integer`).

Usage

```
## S4 method for signature 'character,character'
svattr(S, V)

## S4 method for signature 'numeric,numeric'
svattr(S, V)

## S4 method for signature 'integer,integer'
svattr(S, V)
```

Arguments

S	the *summary* value
V	the *variation* value

Slots

S either character or numeric or integer of length one

V either character or numeric or integer of length one

Methods

Slot accessors

summary(object, ...) Gets the value from the S slot

variation(object, ...) Gets the value from the V slot

Note

The slots S and V must be of the same class (either character, numeric, or integer).

SVTable-class *Summary-Variation Table*

Description

The SVTable class stores a matrix composed by two columns: S (for *summary*) and V (for *variation*).

This class is inherited by GEVASummary.

Slots

sv matrix composed by two columns: S (summary) and V (variation)

Methods

Constructor

svtable(S, V, row.names = NULL) Creates a SVTable from the vectors S and V

Dimension accessors

dim(x) Gets the dimensions from the sv slot

dimnames(x) Gets a list with the row and column names from the sv slot.
Individual dimension names can also be accessed through rownames and colnames

Plotting

points(x, which, ..., classif) Draws the SV points in the plot

Slot accessors

`sv(object)` Gets the `matrix` from the `sv` slot

Note

The matrix from `sv` slot can `numeric`, `character`, or any other supported type by `matrix`. The same slot from `GEVASummary`, however, is always a numeric `matrix`.

`top.genes` *Top Results from GEVA*

Description

Extracts the genes with a relevant classification according to the GEVA results.

Usage

```
top.genes(
  gevaresults,
  classif = c("similar", "factor-dependent", "factor-specific"),
  which.spec = levels(gevaresults),
  add.cols = NULL,
  ...,
  names.only = FALSE
)
```

Arguments

`gevaresults`	a `GEVAResults` object
`classif`	`character` vector, filters the returned genes by their final classification. Possible options are `"similar"`, `"factor-dependent"`, `"factor-specific"`, `"sparse"`, and `"basal"`. Multiple options can be combined
`which.spec`	`factor`, filters the specific factors to be returned
`add.cols`	`character` vector with column names from the feature table (accessed by `featureTable(gevaresul` The matching columns will be added to the returned table
`...`	optional arguments (not used in this version)
`names.only`	`logical`, set to `TRUE` to return only the table row names

Value

If `names.only` is `FALSE` (the default), returns a subset of the `resultstable` slot (`data.frame`) from the `gevaresults` that includes only the filtered genes according to the function parameters.

Otherwise, if `names.only` is `TRUE`, returns only the row names (`character` vector) of this table subset.

Examples

```
## Basic usage with a random generated input
ginput <- geva.ideal.example() # Generates a random input example
gresults <- geva.quick(ginput) # Performs the entire analysis (default parameters)

# Gets a table that includes all the top genes
dtgenes <- top.genes(gresults) # Gets the top genes table
head(dtgenes)                  # Prints the first results

# Appends the "Symbol" column to the results table
dtgenes <- top.genes(gresults, add.cols="Symbol")
head(dtgenes)                  # Prints the first results

# Appends all feature columns to the results table
dtgenes <- top.genes(gresults, add.cols=names(featureTable(gresults)))
head(dtgenes)                  # Prints the first results

# Gets only the factor-specific genes
dtgenes <- top.genes(gresults, "factor-specific")
head(dtgenes)                  # Prints the first results

# Gets only the factor-specific genes for "Cond_1" factor (if any)
dtgenes <- top.genes(gresults, "factor-specific", "Cond_1")
head(dtgenes)                  # Prints the first results
```

`TypedList-class` *Type-strict List (TypedList-class)*

Description

List containing elements of the same class or inheritance.

Slots

`elem.class` `character` representing the class related to the elements

Methods

`typed.list(..., elem.class = NA_character_)` Creates a TypedList from the elements in `...` derived from the class `elem.class`

Slot accessors

`elem.class(typedlist)` Gets the `character` from the `elem.class` slot

`elem.class(typedlist) <- value` Sets a value to the `elem.class` slot