eisa

October 25, 2011

`ALLModules` *ISA transcription modules for the ALL data*

Description

The Iterative Signature Algorithm (ISA) is a biclustering method. `ALLModules` and `ALLModulesSmall` are example ISA biclusters (=modules) found in the ALL data set.

Usage

```
ALLModules
ALLModulesSmall
```

Format

Both `ALLModules` and `ALLModulesSmall` are instances of the `ISAModules` class.

Source

`ISAModules` was generated by calling `ISA` on the `ALL` data set, using the default parameters. `ISAModulesSmall` was generated the same way, but with gene threshold 2.7 and condition threshold 1.4 only.

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

See Also

The `ALL` BioConductor package.

Examples

```
data(ALLModules)
ALLModules
```

ISACHR *Calculate chromosome enrichment for transcription modules*

Description

Hypergeometric test(s) to check whether significantly many genes of an ISA module are on the same chromosome.

Usage

```
ISACHR (modules, ann = annotation(modules), features = featureNames(modules),
    hgCutoff = 0.05, correction = TRUE, correction.method = "holm")
```

Arguments

modules	An ISAModules object, a set of ISA modules.
ann	Character scalar. The annotation package to be used. By default it is taken from the modules argument.
features	Character vector. The names of the features. By default it is taken from the modules argument.
hgCutoff	Numeric scalar. The cutoff value to be used for the enrichment significance. This can be changed later, without recalculating the test.
correction	Logical scalar, whether to perform multiple hypothesis testing correction.
correction.method	Character scalar, the multiple testing correction method to use. Possible values: "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". See the p.adjust function for details on these.

Details

The hypergeometric test, a version Fisher's exact test, takes a chromosome and a gene set (in our case coming from an ISA module) and asks whether the number of genes in the set that are on the given chromosome is significantly more (or less) than what one would expect by chance.

ISACHR performs the hypergeometric test for every module, for every chromosome. The chromosome mapping is taken from the annotation package of the chip.

ISACHR currently cannot test for under-representation.

Value

A CHRListHyperGResult object.

Author(s)

Gabor Csardi <Gabor.Csardi@unil.ch>

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

See Also

ISAGO, ISAKEGG and ISAmiRNA for other enrichment calculations.

The Category package.

Examples

```
data(ALLModulesSmall)
CHR <- ISACHR(ALLModulesSmall)
CHR
sigCategories(CHR)[[2]]
geneIdsByCategory(CHR)[[2]][[1]]
```

ISAGO *Calculate Gene Ontology enrichment for transcription modules*

Description

Gene Ontology enrichment is calculated for each ISA module separately. In the end the result is corrected for multiple hypothesis testing.

Usage

```
ISAGO (modules, ann = annotation(modules), features = featureNames(modules),
    hgCutoff = 0.05, correction = TRUE, correction.method = "holm")
```

Arguments

modules	An ISAModules object, a set of ISA modules.
ann	Character scalar. The annotation package to be used. By default it is taken from the modules argument.
features	Character vector. The names of the features. By default it is taken from the modules argument.
hgCutoff	Numeric scalar. The cutoff value to be used for the enrichment significance. This can be changed later, without recalculating the test.
correction	Logical scalar, whether to perform multiple hypothesis testing correction.
correction.method	Character scalar, the multiple testing correction method to use. Possible values: "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". See the p.adjust function for details on these.

Details

The Gene Ontology is a database of gene annotation. The annotating labels (these are called terms) are standardized and organized into a directed acyclic graph. In other words terms may have more specific sub-terms, that can have even more specific sub-sub-terms, and so on.

The Gene Ontology database has three big sub-graphs, the root nodes (the most general terms) of these are the direct children of the root term of the whole ontology: biological process, cellular component, molecular function. They are usually referred to as ontologies.

The hypergeometric test, a version Fisher's exact test, takes a GO term and a gene set (in our case coming from an ISA module) and asks whether the number of genes in the set annotated by the term is significantly more (or less) than what one would expect by chance.

`ISAGO` performs the hypergeometric test for every module, for all GO terms of the three GO ontologies. The GO data is taken from the `GO.db` package and the annotation package of the chip.

`ISAGO` currently cannot test for under-representation and the conditional test, as implemented in the `GOstats` package, is not available either.

Value

A list with three `GOListHyperGResult` objects, for the three Gene Ontologies, named

`BP`	aka Biological Processes
`CC`	aka Cellular Components
`MF`	aka Molecular Function

Author(s)

Gabor Csardi `<Gabor.Csardi@unil.ch>`

References

The Gene Ontology Consortium. Gene ontology: tool for the unification of biology. *Nat. Genet.* May 2000;25(1):25-9.

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

See Also

`ISAKEGG`, `ISACHR`, `ISAmiRNA` for other enrichment calculations.

The `GO.db`, `GOstats` and `Category` packages.

Examples

```
data(ALLModulesSmall)
GO <- ISAGO(ALLModulesSmall)
GO
summary(GO$BP)[[1]][,1:5]
```

`ISAKEGG` *Calculate KEGG Pathway enrichment for transcription modules*

Description

KEGG pathway enrichment is calculated for each ISA module separately. In the end the result is corrected for multiple hypothesis testing.

Usage

```
ISAKEGG (modules,ann = annotation(modules), features = featureNames(modules),
    hgCutoff = 0.05, correction = TRUE, correction.method = "holm")
```

Arguments

modules	An ISAModules object, a set of ISA modules.
ann	Character scalar. The annotation package to be used. By default it is taken from the modules argument.
features	Character vector. The names of the features. By default it is taken from the modules argument.
hgCutoff	Numeric scalar. The cutoff value to be used for the enrichment significance. This can be changed later, without recalculating the test.
correction	Logical scalar, whether to perform multiple hypothesis testing correction.
correction.method	Character scalar, the multiple testing correction method to use. Possible values: "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". See the p.adjust function for details on these.

Details

KEGG (Kyoto Encyclopedia of Genes and Genomes) is a collection of online databases dealing with genomes, enzymatic pathways, and biological chemicals. The PATHWAY database records networks of molecular interactions in the cells, and variants of them specific to particular organisms.

The hypergeometric test, a version Fisher's exact test, takes a KEGG pathway and a gene set (in our case coming from an ISA module) and asks whether the number of genes in the set participating in the pathway, is significantly more (or less) than what one would expect by chance.

ISAKEGG performs the hypergeometric test for every module, for all KEGG pathways. The KEGG data is taken from the KEGG.db package and the annotation package of the chip.

ISAKEGG currently cannot test for under-representation.

Value

A KEGGListHyperGResult object.

Author(s)

Gabor Csardi <Gabor.Csardi@unil.ch>

References

http://www.genome.jp/kegg/

Kanehisa M, Goto S, Kawashima S, Okuno Y, Hattori M., The KEGG resource for deciphering the genome, *Nucleic Acids Res.* 2004 Jan 1;32(Database issue):D277-80.

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

See Also

ISAGO, ISACHR, ISAmiRNA for other enrichment calculations.

The KEGG.db and Category packages.

Examples

```
data(ALLModulesSmall)
KEGG <- ISAKEGG(ALLModulesSmall)
KEGG
sigCategories(KEGG)[[1]]
summary(KEGG)[[1]][,1:5]
```

ISA *Iterative Signature Algorithm on Gene Expression data*

Description

Run ISA on an ExpressionSet with the default parameters.

Usage

```
ISA (data, flist = filterfun(function(x) IQR(x) > 0.5),
    uniqueEntrez = TRUE, thr.gene = seq(2, 4, by = 0.5),
    thr.cond = seq(1, 3, by = 0.5), no.seeds = 100)
```

Arguments

`data`	The input, an ExpressionSet object.
`flist`	A 'list' of filter functions to apply to the array. This is passed to the `genefilter` function without touching it. Supply `NA` here if you don't want to filter the expression set before running ISA on it.
`uniqueEntrez`	Logical scalar, whether to filter the input expression set to keep exactly one probeset for each Entrez gene. Probesets that are not mapped to an Entrez gene are dropped.
`thr.gene`	Numeric vector. The threshold parameters for the ISA, for features (=probesets or genes). All combinations of `thr.gene` and `thr.cond` will be used to run ISA.
`thr.cond`	Numeric vector. The threshold parameters for the ISA, for samples. All combinations of `thr.gene` and `thr.cond` will be used to run ISA.
`no.seeds`	Number of seeds to run ISA from.

Details

Please read tutorial vignette included in this package for an introduction on ISA. The `isa2-package` manual page in the `isa2` package is also useful.

The `ISA` function performs the ISA algorithm on the supplied expression data. This involves the following steps:

1. Filtering the features (i.e. probe sets) according to their variance. You will need the `genefilter` package for this. The default filtering function keeps the features that have an `IQR` of 0.5 or more. See `genefilter` for details on how to create filtering functions. If `NA` is given as the `flist` argument, then no filtering is performed.
2. Filtering the features by mapping them to Entrez genes. Features that do not map to Entrez genes are removed from the data set. If more features map to the same Entrez gene, then only the one with the highest variance will be kept.

3. Calling the `isa` function in the `isa2` package to perform the Iterative Signature Algorithm. This itself performs the following steps:
   (a) Normalizing the data by calling `isa.normalize`.
   (b) Generating random input seeds via `generate.seeds`.
   (c) Running ISA with all combinations of given feature and sample thresholds, by calling `isa.iterate`.
   (d) Merging similar modules, separately for each threshold combination, by calling `isa.unique`.
   (e) Filtering the modules separately for each threshold combination, by calling `isa.filter.robust` in the `isa2` package.
   (f) Putting all modules from the runs with different thresholds into a single object.
   (g) Merging similar modules, across all threshold combinations, if two modules are similar, then the one with the milder thresholds is kept.
4. Creates an `ISAModules` object from the ISA results.

Value

An `ISAModules-class` object.

Author(s)

Gabor Csardi `<Gabor.Csardi@unil.ch>`

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

Ihmels J, Friedlander G, Bergmann S, Sarig O, Ziv Y, Barkai N: Revealing modular organization in the yeast transcriptional network *Nat Genet.* 2002 Aug;31(4):370-7. Epub 2002 Jul 22

Ihmels J, Bergmann S, Barkai N: Defining transcription modules using large-scale gene expression data *Bioinformatics* 2004 Sep 1;20(13):1993-2003. Epub 2004 Mar 25.

See Also

The vignette included in the `eisa` package.

Examples

```
library(ALL)
data(ALL)
modules <- ISA(ALL, thr.gene=2.7, thr.cond=1.4)
modules
```

ISAHTML *Create HTML summary pages from the result of modular analysis*

Description

These functions create various sophisticated HTML pages from a set of ISA biclusters.

Usage

```
ISAHTMLTable (modules, target.dir, which = seq_len(length(modules)),
    template = system.file("autogen", package = "eisa"), GO = NULL,
    KEGG = NULL, miRNA = NULL, CHR = NULL, htmltitle = NULL,
    notes = NULL, seed = NULL)

ISAHTMLModules (eset, modules, which = seq_len(length(modules)), target.dir,
    template = system.file("autogen", package = "eisa"), GO = NULL,
    KEGG = NULL, miRNA = NULL, CHR = NULL, cond.to.include = NULL,
    cond.col = "white", sep = NULL, seed = NULL, condPlot = TRUE)

ISAHTML (eset, modules, target.dir, template = system.file("autogen",
    package = "eisa"), GO, KEGG, miRNA = NULL, CHR = NULL, htmltitle = NULL,
    notes = NULL, seed = NULL, cond.to.include = NULL, cond.col = "white",
    sep = NULL, condPlot = TRUE)
```

Arguments

modules	An ISAModules object.
target.dir	Character vector of length one, the directory in which the result is placed. It is created if it does not exist.
which	Numeric vector, which modules to include in the table (for ISAHTMLTable); or, which modules to create HTML pages for (ISAHTMLModules). All modules are used by default.
template	The directory containing the HTML template files. By default the template included in the eisa package is used.
GO	List of three GOListHyperGResult objects, as returned by the ISAGO function.
KEGG	A KEGGListHyperGResult object, usually the output of the ISAKEGG function.
miRNA	A miRNAListHyperGResult object, or NULL. See also the ISAmiRNA function.
CHR	A CHRListHyperGResult object or NULL, see also the ISACHR function.
htmltitle	Character vector of length one, the title of the HTML page.
notes	Character vector of length one. Optional HTML text, on the default template it is placed on the top of the page, above the table.
seed	Either NULL, or a character vector, with an optional column that is added to the module table.
eset	An ExpressionSet or ISAExpressionSet object. If an ExpressionSet object is supplied, then it is normalised by calling ISANormalize on it.

`cond.to.include`
: Numeric or character vector, specifies which columns of the phyenotype data of the original expression matrix are included in the tables of samples. By default the first six columns are included.

`cond.col`
: This is passed to `condPlot` as the `col` argument.

`sep`
: This is passed to `condPlot` as the `sep` argument.

`condPlot`
: Logical scalar, whether to create condition plots. If an alternative biclustering method was used to find the modules, then probably it makes no sense creating condition plots for them.

Details

`ISAHTMLTable` creates an HTML page, a summary of the results of the modular analysis, including enrichment analysis of the modules.

`ISAHTMLModules` creates a separate HTML page for each module, including the following elements:

- An expression plot of the genes and samples in the module, including the ISA scores. This is done by calling `expPlot`.
- Gene Ontology tree plots for the enriched GO terms, separately for the three ontologies. These are produced by calling `gograph`.
- Tables for the enriched Gene Ontology terms, separately for the three ontologies.
- A table for the enriched KEGG pathways.
- A table for the enriched miRNA families.
- The list of genes in the module.
- The list of samples in the module.
- A condition plot (if the `condPlot` argument is `TRUE`), see `condPlot`.

By default, clicking on the rows of the table generated by `ISAHTMLTable` is linked to the HTML page of the module, generated by `ISAHTMLModules`.

`ISAHTML` calls both `ISAHTMLTable` and `ISAHTMLModules`.

Value

These functions do not return a value. (They return `NULL`, invisibly.)

Author(s)

Gabor Csardi `<Gabor.Csardi@unil.ch>`

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

See Also

The vignette included in the `eisa` package.

Examples

```
# Load data
library(ALL)
data(ALL)
data(ALLModulesSmall)

# Calculate enrichment
GO <- ISAGO(ALLModulesSmall)
KEGG <- ISAKEGG(ALLModulesSmall)
CHR <- ISACHR(ALLModulesSmall)

# Generate HTML summary
htmldir <- tempdir()
ISAHTML(ALL, modules=ALLModulesSmall, target.dir=htmldir,
        GO=GO, KEGG=KEGG, CHR=CHR)

# Open a browser to view the summary
if (interactive()) {
  browseURL(URLencode(paste("file://", htmldir, "/maintable.html", sep="")))
}
```

ISAIterate *Perform the Iterative Signature Algorithm*

Description

`ISAIterate` performs the ISA on an `ExpressionSet` object, from the given input seeds.

Usage

```
ISAIterate(data, feature.seeds, sample.seeds, thr.feat,
    thr.samp = thr.feat, ...)
```

Arguments

`data`	An `ExpressionSet` or `ISAExpressionSet` object. If an `ExpressionSet` object is supplied, then it is normalised by calling `ISANormalize` on it.
`feature.seeds`	A matrix of feature seeds. The number of rows should match the number of features in the `ExpressionSet`, each column is a seed. Either this, or the `sample.seeds` argument must be given.
`sample.seeds`	A matrix of sample seeds. The number of rows should match the number of samples in the `ExpressionSet`, each column in a seed. Either this, or the `feature.seeds` argument must be given.
`thr.feat`	Numeric scalar or vector giving the threshold parameter for the features. Higher values indicate a more stringent threshold and the result biclusters will contain less features on average. The threshold is measured by the number of standard deviations from the mean, over the values of the feature vector. If it is a vector, then it must contain an entry for each seed.
`thr.samp`	Numeric scalar or vector giving the threshold parameter for the columns. The analogue of `thr.feat`.
`...`	Additional arguments, these are passed to the `isa.iterate` function in the `isa2` package. See also details below.

Details

Performs the ISA from the given seeds. It is allowed to specify both type of seeds, then a half-iteration is performed on the `sample.seeds` and they are appended to the `feature.seeds`.

The `isa.iterate` function of the `isa2` package is called to do all the work, this has the following extra parameters: `direction`, `convergence`, `cor.limit`, `eps`, `corx`, `oscillation`, `maxiter`. Please see the `isa.iterate` manual for details about them.

Value

An `ISAModules` object.

Author(s)

Gabor Csardi `<Gabor.Csardi@unil.ch>`

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

See Also

The `ISA` function for an easier interface with parameters.

Examples

```
library(ALL)
data(ALL)

# Only use a small sample, to make this example finish faster
ALL.normed <- ISANormalize(ALL)[sample(1:nrow(ALL), 1000),]

# Generate seeds and do ISA
seeds <- generate.seeds(nrow(ALL.normed), count=100)
modules <- ISAIterate(ALL.normed, seeds, thr.feat=3, thr.samp=2)
modules
```

ISAmiRNA — *Calculate (predicted) miRNA target enrichment for transcription modules*

Description

This function performs enrichment calculations with respect to predicted miRNA targets to check whether an ISA module contains many genes that are targets of the same miRNA.

Usage

```
ISAmiRNA (modules, ann = annotation(modules), features = featureNames(modules),
    hgCutoff = 0.05, correction = TRUE, correction.method = "holm")
```

Arguments

modules	An ISAModules object, a set of ISA modules.
ann	Character scalar. The annotation package to be used. By default it is taken from the modules argument.
features	Character vector. The names of the features. By default it is taken from the modules argument.
hgCutoff	Numeric scalar. The cutoff value to be used for the enrichment significance. This can be changed later, without recalculating the test.
correction	Logical scalar, whether to perform multiple hypothesis testing correction.
correction.method	Character scalar, the multiple testing correction method to use. Possible values: "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". See the p.adjust function for details on these.

Details

miRNAs are short RNA fragments that specifically regulate (usually inhibit) the expression of genes. Some genes have been experimentally validated as targets of a given miRNA, but we currently don't know the target genes of most miRNAs.

TargetScan is a database of predicted miRNA targets. The predictions are done based many factors, including the conservation of the target region during evolution.

The hypergeometric test, a version Fisher's exact test, takes a miRNA and a gene set (in our case coming from an ISA module) and asks whether the number of genes in the set regulated by the miRNA is significantly more (or less) than what one would expect by chance.

ISAmiRNA performs the hypergeometric test for every module, for all miRNAs in the TargetScan database.

In order to use this function, TargetScan annotation packages are needed. These are currently available for Homo Sapiens and Mus Musculus and they can be downloaded from http://www.unil.ch/cbg/index.php?title=Software.

Value

A miRNAListHyperGResult object.

Author(s)

Gabor Csardi <Gabor.Csardi@unil.ch>

References

Conserved Seed Pairing, Often Flanked by Adenosines, Indicates that Thousands of Human Genes are MicroRNA Targets Benjamin P Lewis, Christopher B Burge, David P Bartel. *Cell*, 120:15-20 (2005).

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

See Also

ISAGO, ISAKEGG and ISACHR for other enrichment calculations.

The Category package.

Examples

```
data(ALLModulesSmall)

if (require(targetscan.Hs.eg.db)) {
  miRNA <- ISAmiRNA(ALLModulesSmall)
  summary(miRNA, p=0.1)[[7]]
}
```

`ISANormalize` *Normalize expression data for the Iterative Signature Algorithm*

Description

ISA works best if the input data is centered and scaled. `ISANormalize` performs this transformation.

Usage

```
ISANormalize (data, prenormalize = FALSE)
```

Arguments

`data`	An `ExpressionSet` object.
`prenormalize`	If this argument is set to `TRUE`, then feature-wise scaling is calculated on the sample-wise scaled matrix and not on the input matrix directly.

Details

It was observed that the ISA works better if the input matrix is scaled and its rows have mean zero and standard deviation one.

An ISA step consists of two sub-steps, and this implies two different normalizations, in the first the rows (=features), in the second the columns (=samples) of the input matrix will be scaled and centered.

Value

An `ISAExpressionSet` object.

Author(s)

Gabor Csardi `<Gabor.Csardi@unil.ch>`

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

See Also

The `ISA` function for an easier ISA workflow.

Examples

```
library(ALL)
data(ALL)

# Do the normalization
ALL.normed <- ISANormalize(ALL)
class(ALL.normed)
dim(exprs(ALL.normed))
dim(featExprs(ALL.normed))
dim(sampExprs(ALL.normed))

# Check that we indeed have Z-scores
all(abs(apply(featExprs(ALL.normed), 2, mean) ) < 1e-12)
all(abs(1-apply(featExprs(ALL.normed), 2, sd)) < 1e-12)

all(abs(apply(sampExprs(ALL.normed), 1, mean) ) < 1e-12)
all(abs(1-apply(sampExprs(ALL.normed), 1, sd)) < 1e-12)
```

ISASweep *Create an ISA module tree*

Description

These functions create and plot the hierarchical description of an expression data set, by applying the ISA with various thresholds, and connecting the related modules. See details below.

Usage

```
ISASweep (expset, modules, ...)
ISASweepGraph (sweep.result)
ISASweepGraphPlot (graph, vertex.label=V(graph)$id,
  vertex.label.topleft=NA, vertex.label.topright=NA,
  vertex.label.bottomleft=NA, vertex.label.bottomright=NA,
  vertex.label.cex=0.8, edge.label=NA, asp=FALSE, rescale=FALSE,
  xlim=range(graph$layout[,1]), ylim=range(graph$layout[,2]),
  thresholds=TRUE, xlab=NA, ylab=NA, ...)
```

Arguments

`expset`	The expression set object, if it is not an `ISAExpressionSet`, then `ISANormalize` is called on it.
`modules`	An `ISAModules` object.
`...`	Additional arguments. `ISASweep` passes these to `isa.sweep`; `ISASweepGraphPlot` passes additional arguments to `plot.igraph`.
`sweep.result`	An `ISAModules` object that contains the sweep tree information as well.
`graph`	An igraph graph object, the sweep tree.
`vertex.label`	Vertex labels, by default the ids of the modules.
`vertex.label.topleft`	Vertex labels to put at the top left corner.

`vertex.label.topright`
: Vertex labels to put at the top right corner.

`vertex.label.bottomleft`
: Vertex labels to put at the bottom left corner.

`vertex.label.bottomright`
: Vertex labels to put at the bottom right corner.

`vertex.label.cex`
: Magnification factor for the vertex labels.

`edge.label` Edge labels.

`asp` Logical scalar, whether the plot should have 1:1 aspect ratio.

`rescale` Logical scalar, whether to rescale the layout coordinates to the [-1,1] interval.

`xlim` Numeric vector of length two, the X limits of the plot.

`ylim` Numeric vector of length two, the Y limits of the plot.

`thresholds` Logical scalar, whether to add the (non-constant) thresholds to the plot.

`xlab` The label of the horizontal axis, by default omitted.

`ylab` The label of the vertical axis, by default omitted.

Details

The ISA uses two threshold parameters that tune the sizes of the transcription modules. The sweep graph of an expression set is defined as the following. It is a directed graph, where the vertices are ISA modules, found at some threshold parameter values. There is an edge from module A to module B, if using 1) (the genes of) module A as the seed vector and 2) the threshold parameters used to find module B, the ISA converges to module B.

The `ISASweep` function creates an ISA sweep tree, in which one threshold parameter is kept fixed and the other varies. It starts from the modules found at the most stringent (=highest) threshold parameters, and uses them individually as seeds at the next less stringent threshold level. If this ISA iteration converges to an already known module, then an edge of the sweep tree is found. If the iteration converges to a new module, then this is added to the module list, together with the sweep tree edge. Then we proceed with the next level of modules, towards the less stringent threshold parameters.

The `ISASweepGraph` function creates a graph object that corresponds to the sweep tree of the expression set.

The `ISASweepGraphPlot` function plots a graph created with `ISASweepGraph`.

Value

`ISASweep` returns an `ISAModules` object, with some seed data added.

`ISASweepGraph` returns an igraph graph object.

`ISASweepGraphPlot` returns `NULL`, invisibly.

Author(s)

Gabor Csardi `<Gabor.Csardi@unil.ch>`

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

Examples

```
library(ALL)
data(ALL)

varLimit <- 0.5
kLimit <- 4
ALimit <- 5
flist <- filterfun(function(x) var(x)>varLimit, kOverA(kLimit,ALimit))
ALL.filt <- ALL[genefilter(ALL, flist), ]
ALL.filt2 <- ALL.filt[, grepl("^B", ALL.filt$BT)]

# Run ISA
set.seed(1)
modules <- ISA(ALL.filt2, flist=NA, thr.gene=seq(2,4,by=0.5), thr.cond=1)

# Do the sweep
modules2 <- ISASweep(ALL.filt2, modules)
modules2

# Plot it
G <- ISASweepGraph(modules2)
if (interactive()) {
  ISASweepGraphPlot(G)
}
```

`ISAUnique` *Remove duplicated ISA modules*

Description

From a potentially non-unique set of ISA modules remove all modules that are similar to another module that was found earlier.

Usage

```
ISAUnique(data, isaresult, ...)
```

Arguments

`data`	An `ExpressionSet` or `ISAExpressionSet` object. If an `ExpressionSet` object is supplied, then it is normalised by calling `ISANormalize` on it.
`isaresult`	An `ISAModules` object to be filtered.
`...`	Additional arguments, these are passed to the `isa.unique` function in the `isa2` package. See also details below.

Details

The ISA algorithm might very well find the same modules from many different input seeds, so the output of the `ISAIterate` function is usually not unique: many modules are very similar to each other.

`ISAUnique` eliminates the duplicates and potentially also the non-convergent modules.

The work is performed by calling the `isa.iterate` function in the `isa2` package. The following additional arguments can be specified to be passed to this function:

method Character scalar giving the method to be used to determine if two biclusters are similar. Right now only 'cor' is implemented, this keeps both biclusters if their Pearson correlation is less than cor.limit, both for their row and column scores. See also the neg.cor argument.

ignore.div Logical scalar, if TRUE, then the divergent biclusters will be removed.

cor.limit Numeric scalar, giving the correlation limit for the 'cor' method.

neg.cor Logical scalar, if TRUE, then the 'cor' method considers the absolute value of the correlation.

drop.zero Logical scalar, whether to drop biclusters that have all zero scores.

Value

Another ISAModules object, with unique modules.

Author(s)

Gabor Csardi <Gabor.Csardi@unil.ch>

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

See Also

The ISA function for an easier ISA workflow.

Examples

```
library(ALL)
data(ALL)

# Only use a small sample, to make this example finish faster
ALL.normed <- ISANormalize(ALL)[sample(1:nrow(ALL), 1000),]

# Generate seeds and do ISA
seeds <- generate.seeds(nrow(ALL.normed), count=100)
modules <- ISAIterate(ALL.normed, seeds, thr.feat=3, thr.samp=2)
modules

# Merge the modules
modules2 <- ISAUnique(ALL.normed, modules)
modules2
```

`ISA2heatmap` *Heatmap of a transcription module*

Description

Create a heatmap plot for an ISA module.

Usage

```
ISA2heatmap (modules, module, eset, norm = c("raw", "feature", "sample"),
    scale = c("none", "row", "column"), ...)
```

Arguments

`modules`	An `ISAModules` object.
`module`	Numeric scalar, the number of the module to plot.
`eset`	An `ExpressionSet` or `ISAExpressionSet` object. If an `ExpressionSet` object is supplied (and the `norm` argument is not set to 'raw'), then it is normalised by calling `ISANormalize` on it. A subset of `eset` is selected that corresponds to the features included in `modules`.
`norm`	Character constant, specifies whether and how to normalize the expression values to plot. 'raw' plots the raw expression values, 'feature' the expression values scaled and centered for each feature (=gene) separately and if 'sample' is specified then the expression values are centered and scaled separately for each sample.
`scale`	Character scalar, passed to the `heatmap` function.
`...`	Additional arguments, they are passed to the `heatmap` function.

Value

The same as `heatmap`.

Author(s)

Gabor Csardi `<Gabor.Csardi@unil.ch>`

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

See Also

`heatmap`

Examples

```
library(ALL)
data(ALL)
data(ALLModulesSmall)

if (interactive()) {
  ISA2heatmap(ALLModulesSmall, 1, ALL, norm="feature")
}
```

ISAExpressionSet-class

Expression Set, normalized for using with ISA

Description

An `ExpressionSet` object (`Biobase` package) that contains expression values normalized for use with the Iterative Signature Algorithm.

Usage

```
## S4 method for signature 'ISAExpressionSet'
featExprs(object)
## S4 method for signature 'ISAExpressionSet'
sampExprs(object)

## S4 method for signature 'ISAExpressionSet'
hasNA(object)
## S4 replacement method for signature 'ISAExpressionSet'
hasNA(object) <- value

## S4 method for signature 'ISAExpressionSet'
prenormalized(object)
## S4 replacement method for signature 'ISAExpressionSet'
prenormalized(object) <- value
```

Arguments

`object`	An `ISAExpressionSet` object.
`value`	A logical scalar, new value of the `hasNA` or `prenormalized` attribute.

Details

An `ISAExpressionSet` contains three expression matrices.

In most cases, when then `ISAExpressionSet` was produced by the `ISANormalize` function, these are: the original, raw data, the feature-wise scaled and centered data and the sample-wise scaled and centered data.

Two additional methods were defined to access the extra matrices: `featExprs` returns the feature-wise standardized data, `sampExprs` the sample-wise standardized one.

The `hasNA` function returns `TRUE` if `NA` or `NaN` values appear in at least one of the expression matrices.

The `prenormalized` function returns `TRUE` if the data was prenormalized, see `ISANormalize` for details.

Value

featExprs and sampExprs both return a matrix.

hasNA and prenormalized return a logical vector of length one.

Author(s)

Gabor Csardi <Gabor.Csardi@unil.ch>

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

See Also

ISANormalize, ExpressionSet in the Biobase package.

Examples

```
library(ALL)
data(ALL)

# Do the normalization
ALL.normed <- ISANormalize(ALL)
class(ALL.normed)
dim(exprs(ALL.normed))
dim(featExprs(ALL.normed))
dim(sampExprs(ALL.normed))

# Check that we indeed have Z-scores
all(abs(apply(featExprs(ALL.normed), 2, mean) ) < 1e-12)
all(abs(1-apply(featExprs(ALL.normed), 2, sd)) < 1e-12)

all(abs(apply(sampExprs(ALL.normed), 1, mean) ) < 1e-12)
all(abs(1-apply(sampExprs(ALL.normed), 1, sd)) < 1e-12)
```

ISAModules-class *A set of ISA modules*

Description

An ISAModules object stores the results of one ISA run. It contains a set of biclusters (=modules or transcription modules) and some meta information about the ISA run and the input data.

Usage

```
## S4 method for signature 'ISAModules'
dim(x)
## S4 method for signature 'ISAModules'
featureNames(modules)
## S4 method for signature 'ISAModules'
```

```
sampleNames(modules)
## S4 method for signature 'ISAModules'
annotation(modules)
## S4 method for signature 'ISAModules'
getOrganism(modules)
## S4 method for signature 'ISAModules'
pData(modules)

## S4 method for signature 'ISAModules'
seedData(modules)
## S4 method for signature 'ISAModules'
runData(modules)
## S4 method for signature 'ISAModules'
featureThreshold(modules, mods)
## S4 method for signature 'ISAModules'
sampleThreshold(modules, mods)

## S4 method for signature 'ISAModules'
length(x)
## S4 method for signature 'ISAModules'
getNoFeatures(modules, mods)
## S4 method for signature 'ISAModules'
getNoSamples(modules, mods)

## S4 method for signature 'ISAModules'
getFeatures(modules, mods)
## S4 method for signature 'ISAModules'
getSamples(modules, mods)
## S4 method for signature 'ISAModules'
getFeatureNames(modules, mods)
## S4 method for signature 'ISAModules'
getSampleNames(modules, mods)
## S4 method for signature 'ISAModules'
getFeatureScores(modules, mods)
## S4 method for signature 'ISAModules'
getSampleScores(modules, mods)
## S4 method for signature 'ISAModules'
getFeatureMatrix(modules, binary = FALSE,
      sparse = FALSE, mods)
## S4 method for signature 'ISAModules'
getSampleMatrix(modules, binary = FALSE,
      sparse = FALSE, mods)
## S4 method for signature 'ISAModules'
getFullFeatureMatrix(modules, eset, mods)
## S4 method for signature 'ISAModules'
getFullSampleMatrix(modules, eset, mods)

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

Arguments

x,modules	An ISAModules object.
mods	An optional numeric index vector for the modules. If given, the information is only returned only for the specified modules.
binary	Logical scalar. Whether to binarize the feature or sample scores.
sparse	Logical scalar. Whether to return a sparse matrix. The Matrix package is required for sparse matrices.
eset	An ExpressionSet or ISAExpressionSet object. This is needed for calculating the scores of the features/samples that are not in the module. If an ExpressionSet object is supplied, then it is normalised by calling ISANormalize on it.
i	For '[' an index vector for selecting features (=probes, genes). For '[[' an index vector for selecting modules.
j	For '[' an index vector for selecting samples. It is ignored for '[['.
...	Additional indexing arguments, they are not used, just ignored.
drop	This argument is currently not used, just silently ignored.

Details

An ISAModules object contains a set of biclusters, obtained using one run of the Iterative Signature Algorithm.

Various operations are defined such an object, here we document all of them, in several groups.

Value

dim returns a numeric vector of length two. featureNames and sampleNames return a character vector. annotation and getOrganism return a character vector of length one. pData returns a data frame.

seedData returns a data frame, see more below. runData returns a named list, see more below. featureThreshold and sampleThreshold return a numeric vector.

length returns a numeric scalar. getNoFeatures and getNoSamples return a numeric vector.

getFeatures and getSamples return a list of named numeric vectors. getFeatureNames and getSampleNames return a list of character vectors. getFeatureScores and getSampleScores return a list of named numeric vectors. getFeatureMatrix, getSampleMatrix, getFullFeatureMatrix and getFullSampleMatrix return a numeric matrix.

Information about the input data.

dim returns the dimension of the input expression matrix, number of features times number of samples.

featureNames returns a character vector, the names of the features in the original input matrix. I.e. in the input was an ExpressionSet for an Affymetrix array, then the Affymetrix probe IDs are returned.

sampleNames returns a character vector, the names of the samples in the original expression set.

annotation returns a character scalar, the name of the array for the input expression set. More precisely, the annotation slot of the input ExpressionSet is returned, this is the name of the annotation package to use for the ExpressionSet.

`getOrganism` returns the scientific name of the organism for which the input expression data was measures. This is obtained by loading the annotation package of the input `ExpressionSet` object, so that must be installed.

`pData` returns the phenotypic data attached to the input `ExpressionSet` object, in a data frame, samples as rows and various phenotypic variables as columns.

Information about the ISA run

`seedData` returns information about the modules. Each row of the returned data frame corresponds to one module, the columns are various variables:

iterations The number of ISA iterations needed to find the module.

oscillation The length of the oscillation cycle for oscillating modules, zero for others.

thr.row The feature (=gene) threshold used for finding the module.

thr.col The sample (=condition) threshold used for finding the module.

freq The number of times the module was found. This is always one, unless `ISAUnique` was performed.

rob The robustness score of the module. See `ISARobustness` for details.

rob.limit The robustness limit that was used for filtering the modules. As this depends of the feature and sample thresholds, it may be different for different modules.

`runData` returns information about the ISA runs, it is a named list with elements:

annotation The annotation package corresponding to the input expression set.

organism The scientific name of the organism.

direction The `direction` parameter of the ISA. Please see `ISAIterate` for details.

convergence The method to determine ISA convergence, a character scalar. Please see `ISAIterate` for details.

cor.limit Correlation limit for the "`cor`" convergence criterium, see `ISAIterate` for details.

eps Difference limit for the "`eps`" convergence criterium, see `ISAIterate` for details.

corx Size of the time window for the "`corx`" convergence criterium, see `ISAIterate` for details.

maxiter The maximum number of ISA iterations that was allowed.

oscillation Logical, whether oscillating modules were considered during the ISA iteration.

N Numeric scalar, the number of input seeds that were used to find the modules.

unique Logical scalar, whether `ISAUnique` was run on the modules.

prenormalize Logical scalar, whether the input data was prenormalized during ISA normalization, see `ISANormalize`.

hasNA Logical scalar, whether the normalized input data contained some `NA` or `NaN` values.

rob.perms Numeric scalar, the number of times the input data was scrambled when the modules were filtered according to robustness.

Note that some of these might be missing, i.e. `rob.perms` is only present if `ISAFilterRobust` was performed.

`featureThreshold` returns the feature thresholds that were used to find the modules.

`sampleThreshold` returns the sample thresholds that were used to find the modules.

Information about the modules

`length` returns the number of modules.

`getNoFeatures` returns the number of features (=genes) in the input data. The number of features *after* filtering is returned if the input data was filtered.

`getNoSamples` returns the number of samples (=conditions) in the input data.

Retrieve the modules

`getFeatures` returns the indices of the features included in the modules. It returns a list, with one entry for each module. Each entry contains the indices of the features (=genes) in the corresponding module.

`getSamples` does the same as `getFeatures`, but for samples.

`getFeatureNames` is similar to `getFeatures`, but returns feature names instead of feature indices.

`getSampleNames` is similar to `getSamples`, but returns sample names instead of sample indices.

`getFeatureScores` returns the feature scores for the selected modules (all modules by default). It returns a list, with one entry for each module. Each list entry contains the feature scores for one module, in a named numeric vector.

`getSampleScores` is similar to `getFeatureScores`, but for samples and sample scores.

`getFeatureMatrix` returns feature scores for the specified modules (all modules by default) in a matrix form. The number of rows is the number of features and the number of columns is the number of modules requested. It can optionally binarize the values.

`getSampleMatrix` is similar to `getFeatureMatrix`, but for sample scores.

`getFullFeatureMatrix` is similar to `getFeatureMatrix`, but is also calculates scores for the features that were not included in the module. For this it performs one ISA iteration and omits the thresholding step. You need to supply the normalized (or the original) expression data to make this possible.

`getFullSampleMatrix` is the same as `getFullFeatureMatrix`, but for sample scores.

Indexing

A couple of indexing operations were defined to make it easier selecting subsets of modules, features or samples from an `ISAModules` object.

The '`[[`' double bracket indexing operator can be used with a single index vector to select a subset of modules.

The '`[`' single bracket indexing operator can be used to restrict an `ISAModules` object to a subset of features and/or samples. The first index corresponds to features, the second to samples. Indices can be numeric, logical or character vectors, for the latter feature and sample names are used.

Author(s)

Gabor Csardi `<Gabor.Csardi@unil.ch>`

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

See Also

The vignette included in the `eisa` package.

Examples

```
data(ALLModulesSmall)
ALLModulesSmall

length(ALLModulesSmall)
dim(ALLModulesSmall)
annotation(ALLModulesSmall)
getOrganism(ALLModulesSmall)

seedData(ALLModulesSmall)

getNoFeatures(ALLModulesSmall)
getNoSamples(ALLModulesSmall)

getFeatureScores(ALLModulesSmall, 1)[[1]]
```

ListHyperGParams-class

Classes for quick GO/KEGG/CHR/miRNA target or other enrichment

---

Description

These classes extend the `HyperGParams` class from the `Category` package to perform enrichment calculation quickly for multiple gene sets.

Usage

```
## S4 method for signature 'ListHyperGParams'
makeValidParams(object)
## S4 method for signature 'ListHyperGParams'
drive(p)
## S4 replacement method for signature 'ListHyperGParams,logical'
drive(p) <- dri

## S4 method for signature 'GOListHyperGParams'
ontology(object)
## S4 replacement method for signature 'GOListHyperGParams,character'
ontology(object) <- go
## S4 method for signature 'GOListHyperGParams'
conditional(r)
## S4 replacement method for signature 'GOListHyperGParams,logical'
conditional(r) <- cond

## S4 method for signature 'ListHyperGParams'
hyperGTest(p)
```

Arguments

object,p,r	A ListHyperGParams object.
dri	Logical scalar, whether to store the genes that are in the intersection of the specified gene set and the annotation category.
go	Character scalar, the ontology for GO, possible values: 'BP', 'CC', 'MF'.
cond	Logical scalar, whether to perform conditional enrichment calculation. Currently this option is ignored.

Details

The ListHyperGParams abstract class extends HyperGParams and allows to specify a list of gene sets for the enrichment calculation instead of a single set.

ListHyperGParams calculates the enrichment much faster than the original HyperGParams classes in the Category package, especially if the calculation is performed against the same gene universe for many gene sets.

ListHyperGParams is an abstract class, it is not possible to instantiate objects from it. Instead, its various extensions must be used: GOListHyperGParams, KEGGListHyperGParams, CHRListHyperGParams and miRNAListHyperGParams.

The various ListHyperGParams objects can be created with the standard new command, by giving all necessary arguments. Please see the examples below.

Value

makeValidParmas returns another ListHyperGParams instance that has the same class as its arguments'.

ontology returns a character vector of length one.

conditional returns a logical vector of length one.

drive returns a logical vector of length one.

Member functions

Most of these functions are analogous to the ones defined in the Category package, the only difference is that they handle ListHyperGParams objects.

makeValidParams validates ListHyperGParams object, in particular, it removes duplicate genes, both from the gene universe and the specified gene sets; and it also makes sure that all genes in the gene sets are included in the universe.

ontology can be used to query or set the ontology for enrichment calculated against the GO database.

conditional queries or sets whether conditional GO enrichment will be performed. This feature is not implemented yet, see the Category and GOstats packages for a working implementation and more information.

drive queries or sets whether the intersections of the gene sets and the universe are stored in the result object. This information can be calculated later as well, but it is faster to store it at the same time when the hypergeometric test is performed.

Author(s)

Gabor Csardi <Gabor.Csardi@unil.ch>

See Also

Functions for enrichment calculation of ISA modules: ISAGO, ISAKEGG, ISACHR, ISAmiRNA.

Perhaps see also the vignette in the GOstats package.

Examples

```
# GO enrichment, "by hand"
# Load data first
data(ALLModulesSmall)

# Create gene sets
library(hgu95av2.db)
genes <- getFeatureNames(ALLModulesSmall)
entrez <- lapply(genes, function(x) na.omit(unlist(mget(x,
              hgu95av2ENTREZID))))

# Create universe
universe <- na.omit(unlist(mget(featureNames(ALLModulesSmall),
              hgu95av2ENTREZID)))

# Create parameter object
param <- new("GOListHyperGParams", geneIds=entrez, universeGeneIds=universe,
             pvalueCutoff=0.01, drive=FALSE, ontology="BP",
             conditional=FALSE, testDirection="over",
             annotation=annotation(ALLModulesSmall))

# Do the calculation
GOBP <- hyperGTest(param)

# Inspect the result
GOBP
summary(GOBP)[[1]]

# How to create other parameter objects
paramKEGG <- new("KEGGListHyperGParams", geneIds=entrez,
                 universeGeneIds=universe, drive=FALSE,
                 annotation=annotation(ALLModulesSmall))
paramCHR <- new("CHRListHyperGParams", geneIds=entrez,
                 universeGeneIds=universe, drive=FALSE,
                 annotation=annotation(ALLModulesSmall))

# Enrichment with user-supplied categories, we use a list of
# hand-picked genes that are involved in myelin formation
mygenes <- c("YARS", "NFKB2", "NGFR", "CDH1", "NFAT5", "NDRG1", "GAP43",
 "EGR2", "MSN", "ROCK1", "SREBF2", "SOX10", "FIG4", "EGR1", "PIK3R1",
 "CDC42", "EDN3", "EDNRB", "NCAM1", "DHH", "OMG", "PMP22", "LAMA4",
 "MPDZ", "MTMR2", "REL", "S100A1", "ITGA4", "GFAP", "FGF2", "RPSA",
 "CADM1", "CDH19", "DNM2", "PAX3", "SREBF1", "DAG1", "DRP2", "SDC2",
 "MBP", "RELA", "RELB", "JUN", "NAB1", "MOBP", "SKI",  "COL5A2", "RHOA",
 "NFASC", "NEFL", "MPZ", "MAG", "EDNRA", "ERBB4", "LITAF", "MMP2",
 "PLP1", "CDKN1A", "PAK1", "RDX", "GJB1", "LAMA5", "JAM3", "ITGB1",
 "PARD3", "FABP7", "LAMA2", "ERBB3", "CADM4", "FOXO4", "TSPAN31",
 "GPR126", "PTK2", "RAC1", "CDKN2A", "CLDN5", "ID2", "LAMC1", "SOX2",
 "CNTN2", "ERBB2", "NFKB1", "NAB2", "EDN2", "MMP9", "CCND1", "L1CAM",
 "MOG")
```

```
library(org.Hs.eg.db)
myentrez <- na.omit(unlist(mget(mygenes, revmap(org.Hs.egSYMBOL))))
categories <- list(myelin=myentrez)

data(ALLModules)
genes2 <- getFeatureNames(ALLModules)
entrez2 <- lapply(genes2, function(x) na.omit(unlist(mget(x,
             hgu95av2ENTREZID))))

# Create universe
universe2 <- na.omit(unlist(mget(featureNames(ALLModules),
             hgu95av2ENTREZID)))

paramMY <- new("GeneralListHyperGParams", geneIds=entrez2,
              universeGeneIds=universe2, drive=FALSE,
              annotation=annotation(ALLModulesSmall),
              categories=categories)
MY <- hyperGTest(paramMY)
MY
summary(MY)[[1]]
```

`ListHyperGResult-class`

Classes for quick GO/KEGG/CHR/miRNA target or other enrichment

---

Description

These classes extend the `HyperGResult` class from the `Category` package to perform enrichment calculation quickly for multiple gene sets.

Usage

```
## S4 method for signature 'ListHyperGResult'
summary(object, pvalue = pvalueCutoff(object),
  categorySize = NULL)
## S4 method for signature 'ListHyperGResult'
htmlReport(r, file = "", append = FALSE,
  label = "", digits = 3, summary.args = NULL)
## S4 method for signature 'ListHyperGResult'
pvalues(r)
## S4 method for signature 'ListHyperGResult'
sigCategories(r, p)

## S4 method for signature 'ListHyperGResult'
geneCounts(r)
## S4 method for signature 'ListHyperGResult'
expectedCounts(r)
## S4 method for signature 'ListHyperGResult'
oddsRatios(r)
## S4 method for signature 'ListHyperGResult'
universeCounts(r)
## S4 method for signature 'ListHyperGResult'
geneMappedCount(r)
```

```
## S4 method for signature 'ListHyperGResult'
universeMappedCount(r)
## S4 method for signature 'ListHyperGResult'
geneIdsByCategory(r, catids = NULL)

## S4 method for signature 'ListHyperGResult'
geneIdUniverse(r, cond = FALSE)
```

Arguments

object,r	A ListHyperGResult object.
pvalue,p	Numeric vector of length one, the p-value limit, up to which the terms are listed.
categorySize	A numeric vector of length one, or NULL. If not NULL, then it gives the minimum number of annotated genes in the universe, in order to list the term.
file	A file name, or a connection object. The result is written here. If it is "", then the result is written to the standard output. If it is NULL, then the result is not written anywhere. (But it is always returned, invisibly, see below.)
append	Logical scalar, whether to append the HTML code to the given file, or remove its previous contents if it already exists.
label	An HTML label (<A LABEL=""> tag) to add.
digits	The number of digits to use for the numeric columns.
summary.args	A list of arguments to pass to the summary method.
catids	The categories for which the genes are listed. All categories will be listed if this argument is NULL.
cond	Currently not used.

Details

A ListHyperGResult object can store the results of hypergeometric tests, several gene sets against the same universe. ListHyperGRresult is an extension of HyperGResult, as defined in the Category package.

More precisely, ListHyperGResult is an abstract class, it is not possible to instantiate objects from it. Its extensions are be used instead: GOListHyperGResult, KEGGListHyperGResult, CHRListHyperGResult and miRNAListHyperGResult.

Value

pvalues, geneCounts, expectedCounts, oddsRatios and universeCounts return a list of named numeric vectors.

geneMappedCount returns a numeric vector, universeMappedCount returns a numeric vector of length one.

sigCategories returns a list of character vectors.

geneIdsByCategory returns a list of lists of character vectors.

geneIdUniverse returns a list of character vectors.

summary returns a list of data frames with columns: 'Pvalue', 'OddsRatio', 'ExpCount', 'Count', 'Size' and optionally 'drive'.

htmlReport returns a list of chracter vectors, invisibly.

conditional returns a logical vector of length one. ontology returns a character vector of length one.

Member functions

Most of the member functions are analogous to the ones defined for `HyperGResult` in the `Category` package. Usually the only difference is that they return a list of vectors, with one entry for each gene set, instead of just a single vector.

`pvalues` returns the p-values of the hypergeomatric tests. A list is returned, with one numeric vector entry for each input gene set. The p-values for each gene set are ordered according to decreasing significance.

`geneCounts` returns the number of genes from the gene set that are annotated with the given term. This is returned for all input gene sets, in a list.

`expectedCounts` returns the number of genes that are expected to be annotated with the given term, just by chance. This is calculated for all input gene sets, and returned as a list.

`oddsRatios` returns the odds ratios for each term tested, for all gene sets, in a list of numeric vectors.

`universeCounts` returns the number of genes from the universe that are annotated with the given term, for all gene sets, in a list.

`geneMappedCount` gives the size of the gene sets, as used in the algorithm. This can be different than the size of the input gene sets, because of the elimination of duplicates and genes that are not in the universe, before the actual computation.

`universeMappedCount` gives the size of the gene universe, as used in the computation. This can be different than the size given by the user, because duplicates are eliminated before the computation.

`sigCategories` returns the significant terms, at the given p-value threshold, for all gene sets, as a list.

`geneIdsByCategory` returns a list of lists, one entry for each input gene set. Every entry is a list itself and for each tested term it gives the gene ids from the gene set that are annotated with the given term.

`geneIdUniverse` returns a list of character vectors, one for each term that was tested, giving the ids of the genes from the universe that are annotated with that term.

`summary` returns a list of data frames, one for each input gene set. Each data frame has columns: 'Pvalue', 'OddsRatio', 'ExpCount', 'Count', 'Size' and optionally 'drive'. Each row of the data frame corresponds to a tested term.

`htmlReport` creates a HTML summary from a `ListHyperGParams` object. This consists of one table for each input gene get. The summary can be written to a file, but it is also returned in a list of character vectors. There is one list entry for each input gene set, and each element of the character vector corresponds to one line of HTML code. You need the `xtable` package to use this function.

The following functions are defined for `GOListHyperGResult` objects only.

`conditional` returns a logical vector of length one, whether the test was conditional or not. Conditional testing is currently not implemented, please see the `GOstats` package for a working implementation.

`ontology` returns a character vector of length one, the name of the ontology for the GO test.

Author(s)

Gabor Csardi <Gabor.Csardi@gmail.com>

See Also

Functions for enrichment calculation of ISA modules: ISAGO, ISAKEGG, ISACHR, ISAmiRNA, ISAEnrichment.

Perhaps see also the vignette in the GOstats package.

Examples

```
data(ALLModulesSmall)
GO <- ISAGO(ALLModulesSmall)
GO$CC
sigCategories(GO$CC)[[1]]
summary(GO$CC)[[1]][,1:5]
```

---

condPlot *Plot sample scores of a transcription module*

Description

Creates a barplot of sample (=condition) scores, for a given transcription module. See details below.

Usage

```
condPlot (modules, number, eset, col = "white", all = TRUE, sep = NULL,
    sepcol = "grey", val = TRUE, srt = 90, adj.above = c(0, 0.5),
    adj.below = c(1, 0.5), plot.only = seq_len(ncol(eset)), ...)
```

Arguments

modules	An ISAModules object.
number	An integer scalar, the module to plot.
eset	An ExpressionSet or ISAExpressionSet object. This is needed for calculating the scores of the samples that are not in the module, see the all argument. If an ExpressionSet object is supplied, then it is normalised by calling ISANormalize on it.
col	Color of the bars, it it passed to barplot, so it can be any format barplot accepts. E.g. it can be a character vector with different colors for the different bars.
all	Logical scalar, whether to plot all samples (if TRUE, the default), or just the ones that are included in the module.
sep	NULL or a numeric vector. If not NULL, then the bars are separated at the given positions with vertical lines. This is useful if you want to subdivide the samples into groups.
sepcol	The color of the separating lines (see the sep argument), if they are plotted.
val	Logical scalar, whether to add labels with the actual score values.
srt	Numeric scalar, the rotation angle of the text labels, this is passed to the text function.
adj.above	Adjustment of the text labels that are above the bars. This is passed to text, see its manual for details.

adj.below	Adjustments of the text labels that are below the bars. This is passed to text, see its manual for details.
plot.only	Numeric vector, if supplied it is used to plot a subset of samples only. By default all samples are plotted.
...	Additional argument, to be passed to barplot.

Details

condPlot creates a barplot for the sample scores of an ISA transcription module. Each sample is represented as a bar.

These plots are useful if you want to show that a given transcription module separates the samples into two (or more) groups. You can assign different colors to the samples, based on some external information, e.g. case and control samples can be colored differently.

In most cases the scores are between minus one and one, but this is not necessarily true.

It is possible to assign scores to samples that are not part of the module, but this requires performing one (more precisely half) ISA iteration step. Currently the function always performs this extra step, even if the out-of-module samples are not plotted. Because the sample scores in a module are only approximately constant during ISA iteration, it might be possible that the plotted scores are slightly different than the ones stored in the modules variable.

Value

None.

Author(s)

Gabor Csardi <Gabor.Csardi@unil.ch>

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

See Also

ISA and ISAModules.

Examples

```
data(ALLModulesSmall)
library(ALL)
data(ALL)

col <- ifelse(grepl("^B", ALL$BT), "darkolivegreen", "orange")
condPlot(ALLModulesSmall, 1, ALL, col=col)
```

ISA-Biclust conversion

Convert ISA modules to a Biclust object, or the opposite

Description

The biclust package implements several biclustering algorithms in a unified framework. The result of the biclustering is a Biclust object. These functions allow the conversion between Biclust and ISAModules objects.

Usage

```
annotate(biclusters, data)
```

Arguments

biclusters	A Biclust object.
data	An ExpressionSet object.

Details

To convert an ISAModules object (mods) to a Biclust object (bc), you can do:

```
bc <- as(mods, "Biclust")
```

The seed data and run data of the ISAModules object is stored in the Parameters slot of the Biclust object. The ISA scores are binarized by the conversion.

To convert a Biclust object (bc) to an ISAModules object (mods), you can call:

```
mods <- as(bc, "ISAModules")
```

The Parameters slot of the Biclust object is used as the run data of the ISAModules object. The seed data of the new object will be an empty data frame.

The annotate function puts biological annotation into a Biclust object. It is suggested to use it before converting the Biclust object to ISAModules, so that ISA visualization functions and enrichment calculations can make use of this information.

Value

annotate returns a Biclust object.

Author(s)

Gabor Csardi <Gabor.Csardi@unil.ch>

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

Sebastian Kaiser, Rodrigo Santamaria, Roberto Theron, Luis Quintales and Friedrich Leisch. (2009). biclust: BiCluster Algorithms. R package version 0.8.1. http://CRAN.R-project.org/package=biclust

Examples

```
if (require(biclust)) {

  library(ALL)
  data(ALL)
  ALL.filtered <- ALL[sample(1:nrow(ALL), 1000),]

  # Biclust -> ISAModules
  set.seed(1)
  Bc <- biclust(exprs(ALL.filtered), BCPlaid(),
                fit.model = ~m + a + b, verbose = FALSE)
  Bc <- annotate(Bc, ALL.filtered)
  modules <- as(Bc, "ISAModules")
  Bc
  modules
  getNoFeatures(modules)
  getNoSamples(modules)

  # ISAModules -> Biclust
  data(ALLModulesSmall)
  Bc2 <- as(ALLModulesSmall, "Biclust")
  ALLModulesSmall
  getNoFeatures(ALLModulesSmall)
  getNoSamples(ALLModulesSmall)
  Bc2

}
```

`enrichment` *Enrichment analysis for transcription modules, based on user-defined*

Description

This function performs enrichment analysis for each ISA module separately, comparing it to user-defined categories. It is useful to test against other databases and annotations than the Gene Ontology or KEGG pathways.

Usage

```
ISAEnrichment (modules, categories, ann = annotation(modules),
    features = featureNames(modules), hgCutoff = 0.05,
    correction = TRUE, correction.method = "holm")
```

Arguments

modules	An ISAModules object, a set of ISA modules.
categories	A named list of gene categories. The names of the entries are used as category names. Each entry of the list must be a character vector containing Entrez gene ids.
ann	Character scalar. The annotation package to be used. By default it is taken from the modules argument.
features	Character vector. The names of the features. By default it is taken from the modules argument.
hgCutoff	Numeric scalar. The cutoff value to be used for the enrichment significance. This can be changed later, without recalculating the test.
correction	Logical scalar, whether to perform multiple hypothesis testing correction.
correction.method	Character scalar, the multiple testing correction method to use. Possible values: "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". See the p.adjust function for details on these.

Details

This function performs enrichment analysis, based on user defined gene labels. It is useful if one want to test ISA modules against databases, other than GO and KEGG.

The hypergeometric test, a version Fisher's exact test, takes a gene label and a gene set (in our case coming from an ISA module) and asks whether the number of genes in the set labelled by the label is significantly more (or less) than what one would expect by chance.

ISAEnrichment performs the hypergeometric test for every module, for all user supplied gene labels. The mapping from the probe ids on the array to Entrez Ids is done using the appropriate chip annotation package.

ISAEnrichment currently cannot test for under-representation.

Value

A GeneralListHyperGResult object.

Author(s)

Gabor Csardi <Gabor.Csardi@unil.ch>

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

See Also

ISAGO, ISACHR, ISAKEGG and ISAmiRNA for other enrichment calculations.

The Category package.

Examples

```
data(ALLModulesSmall)
library(hgu95av2.db)
entrez <- unique(unlist(mget(featureNames(ALLModulesSmall), hgu95av2ENTREZID)))
categories <- lapply(letters, function(x) sample(entrez, 100))
names(categories) <- letters
fakeEnrichment1 <- ISAEnrichment(ALLModulesSmall, categories, correction=FALSE)
fakeEnrichment2 <- ISAEnrichment(ALLModulesSmall, categories, correction=TRUE)
```

---

expPlot *Expression matrix plots for ISA modules*

Description

These functions create an expression matrix plot for an ISA module. The gene and sample scores are also plotted.

Usage

```
expPlotCreate (eset, modules, which, norm = c("sample", "raw", "feature"))
expPlot (epo, scores = TRUE)
expPlotColbar (epo)
```

Arguments

eset	An ExpressionSet or ISAExpressionSet object. If an ExpressionSet object is supplied (and the norm argument is not set to 'raw'), then it is normalised by calling ISANormalize on it. A subset of eset is selected that corresponds to the features included in modules.
norm	Character constant, specifies whether and how to normalize the expression values to plot. 'raw' plots the raw expression values, 'feature' the expression values scaled and centered for each feature (=gene) separately and if 'sample' is specified then the expression values are centered and scaled separately for each sample.
modules	An ISAModules object.
which	Numeric scalar, which module to plot.
scores	Logical scalar, whether to plot the scores as well.
epo	An object returned by expPlotCreate.

Details

expPlotCreate creates an object that contains all data for performing the image plot and returns it. The reason for not plotting it directly is, that the size of the plot is usually different in different cases, and the opening of the graphics device is delayed until expPlotCreate returns.

In the returned object, the weight and height entries give the optimal size of the image, in pixels.

expPlot creates the expression plot.

expPlotColbar plots a color bar for the expression plot.

Value

`expPlotCreate` returns an `ISAexpPlot` object. It is a named list and has several entries, the important ones:

`width`	Numeric scalar, the optimal width of the plot.
`height`	Numeric scalar, the optimal height of the plot.

`expPlot` returns, invisibly, a named list with members:

`coords`	A list with two entries: `x` and `y`, both numeric vectors of length two. They give the position of the actual expression matrix on the plot.
`gene.width`	Numeric scalar, the width of one box on the image plot, in pixels; if the image size is exactly the suggested one.
`cond.height`	Numeric scalar, the height of one box on the image plot, in pixels; if the image size is exactly the suggested one.

`expPlotColbar` returns `NULL`, invisibly.

Author(s)

Gabor Csardi `<Gabor.Csardi@unil.ch>`

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

See Also

The vignette in the `eisa` package for other ISA visualizations. The `ExpressionView` package for an interactive version.

Examples

```
data(ALLModulesSmall)
library(ALL)
data(ALL)

ep <- expPlotCreate(ALL, ALLModulesSmall, 1)
ep

if (interactive()) {
  expPlot(ep)
}
```

gograph *Plot part of the Gene Ontology hierarchy*

Description

These functions help creating a plot of the Gene Ontology hierarchy.

Usage

```
gograph (table, colbar.length = 30, label.cex = 1, alpha=1, abbrev=5,
    GOGRAPHS = NULL, go.terms = NULL)
gographPlot (graph, coords = FALSE, ...)
```

Arguments

`table`	A data frame with one column, containing the p-values of the enriched GO terms. The row names of the data frame should contain the GO ids.
`colbar.length`	Numeric scalar, the length of the color bar.
`label.cex`	Numeric scalar, factor for the label sizes, e.g. '2' means double size compared to the default.
`alpha`	Alpha channel for the fill color of the vertices.
`abbrev`	Numeric scalar, the minimum length for the abbreviated GO ids.
`GOGRAPHS,go.terms`	These are for internal use only.
`graph`	An `igraph` graph, as returned by the `gograph` function.
`coords`	Logical scalar, whether to return the coordinates of the vertices on the plot.
`...`	Additional arguments. These are passed to `plot.igraph`.

Details

A GO plot can be created in two steps. `gograph` creates an `igraph` graph object that contains all the information about the plot; `gographPlot` creates the actual plot.

The two steps are needed, because `gograph` calculates the optimal size of the plot, and then a graphics device of this size can be created before calling `gographPlot`.

The optimal size is returned by `gograph` in the `width` and `height` graph attributes, these can be queried with

```
G <- gograph(...)
G$width
G$height
```

Value

`gograph` returns an `igraph` object.

`gographPlot` by default returns `NULL`, invisibly. If the `coords` argument is `TRUE`, then it returns the coordinates of the vertices on the plot.

Author(s)

Gabor Csardi <Gabor.Csardi@unil.ch>

References

The Gene Ontology Consortium. Gene ontology: tool for the unification of biology. *Nat. Genet.* May 2000;25(1):25-9.

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

See Also

The `igraph` package for more about `igraph` graphs.

Examples

```
data(ALLModulesSmall)
GO <- ISAGO(ALLModulesSmall)
gotab <- summary(GO$BP)[[1]][,"Pvalue",drop=FALSE]

G <- gograph(gotab)
if (interactive()) {
  x11(width=G$width/15, height=G$height/15)
  gographPlot(G)
}
```

---

mnplot *Plot group means against each other, for an ISA module*

Description

Plot mean expression values for two sets of samples, against each other.

Usage

```
mnplot (x, expset, group, ...)
ISAmnplot (modules, number, eset, norm = c("raw", "feature", "sample"),
     group, ...)
```

Arguments

`x`	A character vector, the feature names for which the plot is created.
`expset`	An `ExpressionSet` object (`Biobase` package), or an expression matrix, with row names as feature names.
`eset`	An `ExpressionSet` or `ISAExpressionSet` object. If an `ExpressionSet` object is supplied (and the `norm` argument is not set to 'raw'), then it is normalised by calling `ISANormalize` on it. A subset of `eset` is selected that corresponds to the features included in `modules`.

norm	Character constant, specifies whether and how to normalize the expression values to plot. 'raw' plots the raw expression values, 'feature' the expression values scaled and centered for each feature (=gene) separately and if 'sample' is specified then the expression values are centered and scaled separately for each sample.
group	A factor that defines two groups to plot one against the other.
modules	An ISAModules object.
number	A numeric scalar, the number of the module for which the plot is created.
...	Additional arguments, they are passed to the plot function.

Details

mnplot plots two group-means against each other, the mean expression of all the specified probes. The two groups are specified as a factor with two levels.

ISAmnplot calls mnplot and plots the mean expression of genes in an ISA module, again, for two groups.

Value

Both functions return invisibly a matrix with two lines, the mean expression values for the two groups, for all the specified genes.

Author(s)

Gabor Csardi <Gabor.Csardi@unil.ch>

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

See Also

The GOmnplot and KEGGmnplot functions in the annotate package.

Examples

```
data(ALLModulesSmall)
library(ALL)
data(ALL)
group <- ifelse(grepl("^B", ALL$BT), "B-cell", "T-cell")
ISAmnplot(ALLModulesSmall, 2, ALL, norm="feature", group=group)
```

`overlap` *Overlap of ISA biclusters*

Description

Plots a network, where each node is a module and modules that overlap are closer to each other.

Usage

```
overlap (modules, algorithm = c("mds", "fr", "drl"), edge.limit = 0.5)
overlapPlot (graph, xsize = 400, ysize = 400, vertex.size = 20,
     vertex.size2 = 10, ...)
```

Arguments

`modules`	An `ISAModules` object.
`algorithm`	The algorithm to use for placing the vertices, a character scalar. See details below.
`edge.limit`	Numeric constant between zero and one, only edges between modules that have a Pearson correlation higher than `edge.limit` will be drawn.
`graph`	An `igraph` object, as returned by `overlap`.
`xsize`	The width of the plot in pixels, only used to calculate the return value, it does not influence the plot itself.
`ysize`	The height of the plot in pixels, only used to calculate the return value, it does not influence the plot itself.
`vertex.size`	The width of the vertices on the plot.
`vertex.size2`	The height of the vertices on the plot.
`...`	Additional arguments, these are passed to the `plot.igraph` function from the `igraph` package.

Details

An `ISAModules` object may potentially contain many modules that overlap. These functions visualize the overlapping relationships of a set of modules.

`overlap` creates an `igraph` graph with additional information on how to plot this graph in a way that nodes representing overlapping modules are close to each other.

`overlapPlot` takes such a graph and plots it.

`overlap` can use various algorithms, depending on the `algorithm` argument. If it is '`mds`', then multi-dimensional scaling is used, by calling the `isaMDS` function in the `MASS` package. If it is '`fr`', then the Fruchterman-Reingold algorithm is used, through the `layout.fruchterman.reingold` function of the `igraph` package. If it is '`drl`', then the DrL graph layout algorithm is used, see the `layout.drl` function in the `igraph` package.

Value

`overlap` returns an igraph graph.

`overlapPlot` returns the coordinates of the vertices in a two-column matrix, invisibly.

Author(s)

Gabor Csardi <Gabor.Csardi@unil.ch>

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

Examples

```
data(ALLModulesSmall)
G <- overlap(ALLModulesSmall, algorithm="drl", edge.limit=0.3)
if (interactive()) {
  overlapPlot(G)
}
```

---

profilePlot *Profile plots for ISA biclusters*

Description

Line plots to compare biclusters to the background, i.e. the rest of the expression matrix.

Usage

```
profilePlot (modules, module, eset, plot = c("samples", "features",
     "both"), norm = "default", background = TRUE,
    col = gray(0.7), col.mod = 1, type = "l", type.mod = type,
    mean = TRUE, meancol = "green", meancol.mod = "red",
    xlabs = c("Features", "Samples"), ylab = "Expression",
    ...)
```

Arguments

modules	An ISAModules object.
module	Numeric scalar, the module to plot.
eset	An ExpressionSet or ISAExpressionSet object. If an ExpressionSet object is supplied (and the norm argument is not set to 'raw'), then it is normalised by calling ISANormalize on it. A subset of eset is selected that corresponds to the features included in modules.
plot	Character constant, specifies what to plot. 'sample' plots sample scores, 'features' plots feature scores. If 'both' is given, then the plot is divided into two subplots and both scores are plotted.
norm	Character constant, specifies how to normalize the expression matrix for plotting. It can be of length one or two, the latter for the case when plots are made both for features and samples. Possible values: 'raw' uses the raw expression values; 'feature' uses featExprs to extract the expression values from the expression set object; 'sample' uses sampExprs; 'default' means 'feature' for sample plots and 'sample' for feature plots.

background	Logical scalar, whether to plot the features/samples that are not in the module.
col	Color of lines corresponding to the background features/samples.
col.mod	Color of the lines corresponding to the features/samples included in the module.
type	Type of the plot, for the background features/samples. It is passed to plot.
type.mod	Type of the plot, for the features/samples included in the module. It is passed to plot.
mean	Logical scalar, whether to plot the mean expression for each feature/sample, separately for the samples/features that are in the module and the ones that are not.
meancol	Color of the line for the mean expression values, background.
meancol.mod	Color of the line for the mean expression values, module.
xlabs	Character vector of length one or two. The labels of the horizontal axes of the plot, the second value is used if both the feature and the sample plots are drawn.
ylab	Character vector of length one. The label of the vertical axes.
...	Additional graphical arguments. They are passed to the lines function that creates the lines of the plot.

Details

plot="both" uses the mfrow graphical parameter to create the two subplots. This does not work properly if you already have subplots.

Value

None. (Well, NULL, invisibly.)

Author(s)

Gabor Csardi <Gabor.Csardi@uni.ch>

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

See Also

The similar parallelCoordinates function in the biclust package.

Examples

```
data(ALLModulesSmall)
library(ALL)
data(ALL)
if (interactive()) {
  profilePlot(ALLModulesSmall, 2, ALL, plot="samples")
}
```

`ISAFilterRobust` *Robustness of ISA biclusters*

Description

Robustness of ISA biclusters. The more robust biclusters are more significant, in the sense that they are less likely to be found in random data.

Usage

```
ISARobustness(data, isaresult)
ISAFilterRobust(data, isaresult, ...)
```

Arguments

`data`	An `ExpressionSet` or `ISAExpressionSet` object. If an `ExpressionSet` object is supplied, then it is normalised by calling `ISANormalize` on it.
`isaresult`	An `ISAModules` object, a set of modules.
`...`	Additional arguments, they are passed to the `isa.filter.robust` function in the `isa2` package.

Details

`ISARobustness` calculates robustness scores for ISA modules. The higher the score, the more robust the module.

`ISAFilterRobust` filters a set of ISA modules, by running ISA on the randomized expression data and then eliminating all modules that have a robustness score that is lower than at least one robustness score found in the randomized data.

The same feature and sample thresholds are used to calculate the randomized robustness scores. In other words the limit for the filtering depends on the feature and sample thresholds.

You can find more details in the manual of the `robustness` function in the `isa2` package.

Value

`ISARobustness` returns a numeric vector, the robustness scores of the biclusters.

`ISAFilterRobust` returns the filtered `ISAModules` instance.

Author(s)

Gabor Csardi `<Gabor.Csardi@unil.ch>`

References

Bergmann S, Ihmels J, Barkai N: Iterative signature algorithm for the analysis of large-scale gene expression data *Phys Rev E Stat Nonlin Soft Matter Phys.* 2003 Mar;67(3 Pt 1):031902. Epub 2003 Mar 11.

See Also

The `robustness` function in the `isa2` package.

Examples

```
data(ALLModules)
library(ALL)
data(ALL)
rob <- ISARobustness(ALL, ALLModules)
summary(rob)
```

Index

∗Topic **classes**
ISAExpressionSet-class, 19
∗Topic **cluster**
condPlot, 31
enrichment, 34
expPlot, 36
gograph, 38
ISA, 6
ISA-Biclust conversion, 33
ISA2heatmap, 18
ISACHR, 2
ISAFilterRobust, 44
ISAGO, 3
ISAHTML, 8
ISAIterate, 10
ISAKEGG, 4
ISAmiRNA, 11
ISAModules-class, 20
ISANormalize, 13
ISASweep, 14
ISAUnique, 16
ListHyperGParams-class, 25
ListHyperGResult-class, 28
mnplot, 39
overlap, 41
profilePlot, 42
∗Topic **datasets**
ALLModules, 1
[,ISAModules,ANY,ANY-method
(ISAModules-class), 20
[,ISAModules,ANY-method
(ISAModules-class), 20
[[,ISAModules,ANY,ANY-method
(ISAModules-class), 20
[[,ISAModules,ANY-method
(ISAModules-class), 20

ALLModules, 1
ALLModulesSmall *(ALLModules)*, 1
annotate *(ISA-Biclust conversion)*, 33
annotation,ISAModules-method
(ISAModules-class), 20
anrichment *(enrichment)*, 34
barplot, *31, 32*
Biclust, *33*

CHRListHyperGParams
(ListHyperGParams-class), 25
CHRListHyperGParams-class
(ListHyperGParams-class), 25
CHRListHyperGResult, *2*, *8*
CHRListHyperGResult
(ListHyperGResult-class), 28
CHRListHyperGResult-class
(ListHyperGResult-class), 28
class:ISAModules
(ISAModules-class), 20
coerce,Biclust,ISAModules-method
(ISA-Biclust conversion), 33
coerce,ISAModules,Biclust-method
(ISA-Biclust conversion), 33
condGeneIdUniverse,CHRListHyperGResult-method
(ListHyperGResult-class), 28
condGeneIdUniverse,GeneralListHyperGResult-met
(ListHyperGResult-class), 28
condGeneIdUniverse,GOListHyperGResult-method
(ListHyperGResult-class), 28
condGeneIdUniverse,KEGGListHyperGResult-method
(ListHyperGResult-class), 28
condGeneIdUniverse,ListHyperGResult-method
(ListHyperGResult-class), 28
condGeneIdUniverse,miRNAListHyperGResult-metho
(ListHyperGResult-class), 28
conditional,GOListHyperGParams-method
(ListHyperGParams-class),

25
conditional,GOListHyperGResult-method (ListHyperGResult-class), 28
conditional<-,GOListHyperGParams,logical-method (ListHyperGParams-class), 25
condPlot, 9, 31

dim,ISAModules-method (ISAModules-class), 20
drive (ListHyperGParams-class), 25
drive,CHRListHyperGParams-method (ListHyperGParams-class), 25
drive,GeneralListHyperGParams-method (ListHyperGParams-class), 25
drive,GOListHyperGParams-method (ListHyperGParams-class), 25
drive,KEGGListHyperGParams-method (ListHyperGParams-class), 25
drive,ListHyperGParams-method (ListHyperGParams-class), 25
drive,miRNAListHyperGParams-method (ListHyperGParams-class), 25
drive<- (ListHyperGParams-class), 25
drive<-,CHRListHyperGParams,logical-method (ListHyperGParams-class), 25
drive<-,GeneralListHyperGParams,logical-method (ListHyperGParams-class), 25
drive<-,GOListHyperGParams,logical-method (ListHyperGParams-class), 25
drive<-,KEGGListHyperGParams,logical-method (ListHyperGParams-class), 25
drive<-,ListHyperGParams,logical-method (ListHyperGParams-class), 25
drive<-,miRNAListHyperGParams,logical-method (ListHyperGParams-class), 25

enrichment, 34

expectedCounts,CHRListHyperGResult-method (ListHyperGResult-class), 28
expectedCounts,GeneralListHyperGResult-method (ListHyperGResult-class), 28
expectedCounts,GOListHyperGResult-method (ListHyperGResult-class), 28
expectedCounts,KEGGListHyperGResult-method (ListHyperGResult-class), 28
expectedCounts,ListHyperGResult-method (ListHyperGResult-class), 28
expectedCounts,miRNAListHyperGResult-method (ListHyperGResult-class), 28
expPlot, 9, 36
expPlotColbar (expPlot), 36
expPlotCreate (expPlot), 36

featExprs, 42
featExprs (ISAExpressionSet-class), 19
featExprs,ISAExpressionSet-method (ISAExpressionSet-class), 19
featureNames,ISAModules-method (ISAModules-class), 20
featureThreshold (ISAModules-class), 20
featureThreshold,ISAModules-method (ISAModules-class), 20

geneCounts,CHRListHyperGResult-method (ListHyperGResult-class), 28
geneCounts,GeneralListHyperGResult-method (ListHyperGResult-class), 28
geneCounts,GOListHyperGResult-method (ListHyperGResult-class), 28
geneCounts,KEGGListHyperGResult-method (ListHyperGResult-class), 28
geneCounts,ListHyperGResult-method (ListHyperGResult-class), 28
geneCounts,miRNAListHyperGResult-method (ListHyperGResult-class),

28
genefilter, *6*
geneIdsByCategory,CHRListHyperGResult-method
(*ListHyperGResult-class*),
28
geneIdsByCategory,GeneralListHyperGResult-method
(*ListHyperGResult-class*),
28
geneIdsByCategory,GOListHyperGResult-method
(*ListHyperGResult-class*),
28
geneIdsByCategory,KEGGListHyperGResult-method
(*ListHyperGResult-class*),
28
geneIdsByCategory,ListHyperGResult-method
(*ListHyperGResult-class*),
28
geneIdsByCategory,miRNAListHyperGResult-method
(*ListHyperGResult-class*),
28
geneIdUniverse,CHRListHyperGResult-method
(*ListHyperGResult-class*),
28
geneIdUniverse,GeneralListHyperGResult-method
(*ListHyperGResult-class*),
28
geneIdUniverse,GOListHyperGResult-method
(*ListHyperGResult-class*),
28
geneIdUniverse,KEGGListHyperGResult-method
(*ListHyperGResult-class*),
28
geneIdUniverse,ListHyperGResult-method
(*ListHyperGResult-class*),
28
geneIdUniverse,miRNAListHyperGResult-method
(*ListHyperGResult-class*),
28
geneMappedCount,CHRListHyperGResult-method
(*ListHyperGResult-class*),
28
geneMappedCount,GeneralListHyperGResult-method
(*ListHyperGResult-class*),
28
geneMappedCount,GOListHyperGResult-method
(*ListHyperGResult-class*),
28
geneMappedCount,KEGGListHyperGResult-method
(*ListHyperGResult-class*),
28
geneMappedCount,ListHyperGResult-method
(*ListHyperGResult-class*),
28
geneMappedCount,miRNAListHyperGResult-method
(*ListHyperGResult-class*),
28
GeneralListHyperGParams
(*ListHyperGParams-class*),
25
GeneralListHyperGParams-class
(*ListHyperGParams-class*),
25
GeneralListHyperGResult, *35*
GeneralListHyperGResult
(*ListHyperGResult-class*),
28
GeneralListHyperGResult-class
(*ListHyperGResult-class*),
28
generateSeeds, *7*
getFeatureMatrix
(*ISAModules-class*), 20
getFeatureMatrix,ISAModules-method
(*ISAModules-class*), 20
getFeatureNames
(*ISAModules-class*), 20
getFeatureNames,ISAModules-method
(*ISAModules-class*), 20
getFeatures (*ISAModules-class*), 20
getFeatures,ISAModules-method
(*ISAModules-class*), 20
getFeatureScores
(*ISAModules-class*), 20
getFeatureScores,ISAModules-method
(*ISAModules-class*), 20
getFullFeatureMatrix
(*ISAModules-class*), 20
getFullFeatureMatrix,ISAModules-method
(*ISAModules-class*), 20
getFullSampleMatrix
(*ISAModules-class*), 20
getFullSampleMatrix,ISAModules-method
(*ISAModules-class*), 20
getNoFeatures (*ISAModules-class*),
20
getNoFeatures,ISAModules-method
(*ISAModules-class*), 20
getNoSamples (*ISAModules-class*),
20
getNoSamples,ISAModules-method
(*ISAModules-class*), 20
getOrganism (*ISAModules-class*), 20
getOrganism,ISAModules-method
(*ISAModules-class*), 20

getSampleMatrix
(ISAModules-class), 20
getSampleMatrix,ISAModules-method
(ISAModules-class), 20
getSampleNames
(ISAModules-class), 20
getSampleNames,ISAModules-method
(ISAModules-class), 20
getSamples (ISAModules-class), 20
getSamples,ISAModules-method
(ISAModules-class), 20
getSampleScores
(ISAModules-class), 20
getSampleScores,ISAModules-method
(ISAModules-class), 20
gograph, 9, 38
gographPlot (gograph), 38
GOListHyperGParams
(ListHyperGParams-class),
25
GOListHyperGParams-class
(ListHyperGParams-class),
25
GOListHyperGResult, 4, 8
GOListHyperGResult
(ListHyperGResult-class),
28
GOListHyperGResult-class
(ListHyperGResult-class),
28
GOmnplot, 40

hasNA (ISAExpressionSet-class), 19
hasNA,ISAExpressionSet-method
(ISAExpressionSet-class),
19
hasNA<- (ISAExpressionSet-class),
19
hasNA<-,ISAExpressionSet-method
(ISAExpressionSet-class),
19
heatmap, 18
htmlReport,CHRListHyperGResult-method
(ListHyperGResult-class),
28
htmlReport,GeneralListHyperGResult-method
(ListHyperGResult-class),
28
htmlReport,GOListHyperGResult-method
(ListHyperGResult-class),
28
htmlReport,KEGGListHyperGResult-method
(ListHyperGResult-class),
28
htmlReport,ListHyperGResult-method
(ListHyperGResult-class),
28
htmlReport,miRNAListHyperGResult-method
(ListHyperGResult-class),
28
hyperGTest,CHRListHyperGParams-method
(ListHyperGParams-class),
25
hyperGTest,GeneralListHyperGParams-method
(ListHyperGParams-class),
25
hyperGTest,GOListHyperGParams-method
(ListHyperGParams-class),
25
hyperGTest,KEGGListHyperGParams-method
(ListHyperGParams-class),
25
hyperGTest,ListHyperGParams-method
(ListHyperGParams-class),
25
hyperGTest,miRNAListHyperGParams-method
(ListHyperGParams-class),
25

IQR, 6
ISA, 6, 11, 13, 17, 32
isa, 7
ISA-Biclust conversion, 33
isa.iterate, 7, 10, 11, 16
isa.normalize, 7
isa.unique, 7, 16
ISA2heatmap, 18
ISACHR, 2, 4, 5, 8, 12, 27, 31, 35
ISAEnrichment, 31
ISAEnrichment (enrichment), 34
ISAExpressionSet, 13
ISAExpressionSet
(ISAExpressionSet-class),
19
ISAExpressionSet-class, 19
ISAFilterRobust, 23, 44
ISAGO, 3, 3, 5, 8, 12, 27, 31, 35
ISAHTML, 8
ISAHTMLModules (ISAHTML), 8
ISAHTMLTable (ISAHTML), 8
ISAIterate, 10, 16, 23
ISAKEGG, 3, 4, 4, 8, 12, 27, 31, 35
ISAmiRNA, 3–5, 8, 11, 27, 31, 35
ISAmnplot (mnplot), 39
ISAModules, 11, 18, 32, 33, 36, 40, 41
ISAModules (ISAModules-class), 20

ISAModules-class, *7*
ISAModules-class, 20
ISANormalize, *8*, *10*, 13, *16*, *18–20*, *22*, *23*, *31*, *36*, *39*, *42*, *44*
ISARobustness, *23*
ISARobustness *(ISAFilterRobust)*, 44
ISASweep, 14
ISASweepGraph *(ISASweep)*, 14
ISASweepGraphPlot *(ISASweep)*, 14
ISAUnique, 16, *23*

KEGGListHyperGParams *(ListHyperGParams-class)*, 25
KEGGListHyperGParams-class *(ListHyperGParams-class)*, 25
KEGGListHyperGResult, *5*, *8*
KEGGListHyperGResult *(ListHyperGResult-class)*, 28
KEGGListHyperGResult-class *(ListHyperGResult-class)*, 28

length,ISAModules-method *(ISAModules-class)*, 20
lines, *43*
ListHyperGParams *(ListHyperGParams-class)*, 25
ListHyperGParams-class, 25
ListHyperGResult *(ListHyperGResult-class)*, 28
ListHyperGResult-class, 28

makeValidParams,CHRListHyperGParams-method *(ListHyperGParams-class)*, 25
makeValidParams,GeneralListHyperGParams-method *(ListHyperGParams-class)*, 25
makeValidParams,GOListHyperGParams-method *(ListHyperGParams-class)*, 25
makeValidParams,KEGGListHyperGParams-method *(ListHyperGParams-class)*, 25
makeValidParams,ListHyperGParams-method *(ListHyperGParams-class)*, 25
makeValidParams,miRNAListHyperGParams-method *(ListHyperGParams-class)*, 25
miRNAListHyperGParams *(ListHyperGParams-class)*, 25
miRNAListHyperGParams-class *(ListHyperGParams-class)*, 25
miRNAListHyperGResult, *8*, *12*
miRNAListHyperGResult *(ListHyperGResult-class)*, 28
miRNAListHyperGResult-class *(ListHyperGResult-class)*, 28
mnplot, 39

oddsRatios,CHRListHyperGResult-method *(ListHyperGResult-class)*, 28
oddsRatios,GeneralListHyperGResult-method *(ListHyperGResult-class)*, 28
oddsRatios,GOListHyperGResult-method *(ListHyperGResult-class)*, 28
oddsRatios,KEGGListHyperGResult-method *(ListHyperGResult-class)*, 28
oddsRatios,ListHyperGResult-method *(ListHyperGResult-class)*, 28
oddsRatios,miRNAListHyperGResult-method *(ListHyperGResult-class)*, 28
ontology,GOListHyperGParams-method *(ListHyperGParams-class)*, 25
ontology,GOListHyperGResult-method *(ListHyperGResult-class)*, 28
ontology<-,GOListHyperGParams,character-method *(ListHyperGParams-class)*, 25
overlap, 41
overlapPlot *(overlap)*, 41

p.adjust, *2*, *3*, *5*, *12*, *35*
parallelCoordinates, *43*
pData,ISAModules-method *(ISAModules-class)*, 20
plot, *40*, *43*

prenormalized
(ISAExpressionSet-class), 19
prenormalized,ISAExpressionSet-method
(ISAExpressionSet-class), 19
prenormalized<-
(ISAExpressionSet-class), 19
prenormalized<-,ISAExpressionSet-method
(ISAExpressionSet-class), 19
print.ISAexpPlot (expPlot), 36
profilePlot, 42
pvalues,CHRListHyperGResult-method
(ListHyperGResult-class), 28
pvalues,GeneralListHyperGResult-method
(ListHyperGResult-class), 28
pvalues,GOListHyperGResult-method
(ListHyperGResult-class), 28
pvalues,KEGGListHyperGResult-method
(ListHyperGResult-class), 28
pvalues,ListHyperGResult-method
(ListHyperGResult-class), 28
pvalues,miRNAListHyperGResult-method
(ListHyperGResult-class), 28

robustness, 44
runData (ISAModules-class), 20
runData,ISAModules-method
(ISAModules-class), 20

sampExprs, 42
sampExprs
(ISAExpressionSet-class), 19
sampExprs,ISAExpressionSet-method
(ISAExpressionSet-class), 19
sampleNames,ISAModules-method
(ISAModules-class), 20
sampleThreshold
(ISAModules-class), 20
sampleThreshold,ISAModules-method
(ISAModules-class), 20
seedData (ISAModules-class), 20
seedData,ISAModules-method
(ISAModules-class), 20
sigCategories,CHRListHyperGResult-method
(ListHyperGResult-class), 28
sigCategories,GeneralListHyperGResult-method
(ListHyperGResult-class), 28
sigCategories,GOListHyperGResult-method
(ListHyperGResult-class), 28
sigCategories,KEGGListHyperGResult-method
(ListHyperGResult-class), 28
sigCategories,ListHyperGResult-method
(ListHyperGResult-class), 28
sigCategories,miRNAListHyperGResult-method
(ListHyperGResult-class), 28
summary,CHRListHyperGResult-method
(ListHyperGResult-class), 28
summary,GeneralListHyperGResult-method
(ListHyperGResult-class), 28
summary,GOListHyperGResult-method
(ListHyperGResult-class), 28
summary,KEGGListHyperGResult-method
(ListHyperGResult-class), 28
summary,ListHyperGResult-method
(ListHyperGResult-class), 28
summary,miRNAListHyperGResult-method
(ListHyperGResult-class), 28

text, 31, 32

universeCounts,CHRListHyperGResult-method
(ListHyperGResult-class), 28
universeCounts,GeneralListHyperGResult-method
(ListHyperGResult-class), 28
universeCounts,GOListHyperGResult-method
(ListHyperGResult-class), 28
universeCounts,KEGGListHyperGResult-method
(ListHyperGResult-class), 28

universeCounts,ListHyperGResult-method
(*ListHyperGResult-class*),
28
universeCounts,miRNAListHyperGResult-method
(*ListHyperGResult-class*),
28
universeMappedCount,CHRListHyperGResult-method
(*ListHyperGResult-class*),
28
universeMappedCount,GeneralListHyperGResult-method
(*ListHyperGResult-class*),
28
universeMappedCount,GOListHyperGResult-method
(*ListHyperGResult-class*),
28
universeMappedCount,KEGGListHyperGResult-method
(*ListHyperGResult-class*),
28
universeMappedCount,ListHyperGResult-method
(*ListHyperGResult-class*),
28
universeMappedCount,miRNAListHyperGResult-method
(*ListHyperGResult-class*),
28