phenoTest

March 24, 2012

`ClusterPhenoTest` *Test association of clusters with phenotype.*

Description

Test the associations between clusters that each sample belongs to (based on gene expression) and each phenotype.

Usage

```
ClusterPhenoTest(x,cluster,vars2test,B=10^4,p.adjust.method='none')
```

Arguments

`x`	ExpressionSet with phenotype information stored in `pData(x)`.
`cluster`	variable of class `character` or `factor` telling at which cluster each sample belongs to.
`vars2test`	list with components 'continuous', 'categorical', 'ordinal' and 'survival' indicating which phenotype variables should be tested. 'continuous', 'categorical' and 'ordinal' must be character vectors, 'survival' a matrix with columns named 'time' and 'event'. The names must match names in `names(pData(x))`.
`B`	An integer specifying the number of replicates used in the chi-square Monte Carlo test (passed on to `chisq.test`).
`p.adjust.method`	Method for P-value adjustment, passed on to `p.adjust`.

Details

Test association between the provided clusters and each phenotype.

For variables in vars2test\$continuous and vars2test\$ordinal a Kruskal-Wallis Rank Sum test is used; for vars2test\$categorical a chi-square test (with exact p-value if `simulate.p.value` is set to TRUE); for var2test\$survival a Cox proportional hazards likelihood-ratio test.

Author(s)

David Rossell

Examples

```
#load data
data(eset)
eset

#construct vars2test
survival <- matrix(c("Relapse","Months2Relapse"),ncol=2,byrow=TRUE)
colnames(survival) <- c('event','time')
#add positive to have more than one category
pData(eset)[1:20,'lymph.node.status'] <- 'positive'
vars2test <- list(survival=survival,categorical='lymph.node.status')
vars2test

#first half of the samples will be one cluster and the rest the other cluster
cluster <- c(rep('Cluster1',floor(ncol(eset)/2)),rep('Cluster2',ncol(eset)-floor(ncol(ese

#test association
ClusterPhenoTest(eset,cluster,vars2test=vars2test)
```

`ExpressionPhenoTest` *Tests univariate association between a list of phenotype variables and gene expression.*

Description

Tests univariate association between a list of phenotype variables and gene expression.

Usage

```
ExpressionPhenoTest(x, vars2test, adjustVars,
p.adjust.method='BH',continuousCategories=3,mc.cores)
```

Arguments

`x`	ExpressionSet containing expression levels in `exprs(x)` and phenotype information in `pData(x)`.
`vars2test`	list with components 'continuous', 'categorical', 'ordinal' and 'survival' indicating which phenotype variables should be tested. 'continuous', 'categorical' and 'ordinal' must be character vectors, 'survival' a matrix with columns named 'time' and 'event'. The names must match names in names(pData(x)).
`adjustVars`	variables that will be used as adjustment variables when fitting linear models and/or cox models. This variables have to exist in `colnames(pData(x))`.
`p.adjust.method`	method for p-value adjustment, passed on to `p.adjust`. Valid values are c("holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none").
`continuousCategories`	number of categories used for continuous variables.
`mc.cores`	the number of cores to use, i.e. how many processes will be spawned (at most).

Details

The effect of both continuous, categorical and ordinal phenotype variables on gene expression levels are tested via lmFit. For ordinal variables a single coefficient is used to test its effect on gene expression (trend test), which is then used to obtain a P-value (means for each category are reported in the output). Gene expression effects on survival are tested via Cox proportional hazards model, as implemented in function 'coxph'.

Value

The output is an `epheno` object, which basically extends an `ExpressionSet` object. The means, fold changes, standarized hazard ratios and pvalues are stored in the `experimentData` slot which is accessible with the `exprs` method. Information about the kind of information of each variable can be found in the `phenoData` slot which is accessible with the `pData` method.

There are several methods that can be used to access the information stored in an `epheno` object. For more information please type one of the following: `getFc(x), getHr(x), getMeans(x), getPvals(x), getSummaryDif(x), logFcHr(x), p.adjust.method(x), phenoClass(x), phenoNames(x).`

Author(s)

David Rossell

Examples

```
#load eset
data(eset)
eset

#prepare vars2test
survival <- matrix(c("Relapse","Months2Relapse"),ncol=2,byrow=TRUE)
colnames(survival) <- c('event','time')
vars2test <- list(survival=survival)

#run ExpressionPhenoTest
epheno <- ExpressionPhenoTest(eset,vars2test,p.adjust.method='none')
epheno
```

---

barplotSignatures-methods

Methods for Function barplotSignatures in Package 'phenoTest'

Description

Methods for function `barplotSignatures` in Package 'phenoTest'. For more information read the function's manual.

Methods

`signature(x = "epheno", signatures = "character")` Method for an `epheno` object and one signature stored in an object of class `character`.

`signature(x = "epheno", signatures = "GeneSetCollection")` Method for an `epheno` object and several signatures stored in an object of class `GeneSetCollection`.

`signature(x = "epheno", signatures = "GeneSet")` Method for an `epheno` object and one signature stored in an object of class `GeneSet`.

`signature(x = "epheno", signatures = "list")` Method for an `epheno` object and several signatures stored in an object of class `list`.

`barplotSignifSignatures-methods`
Methods for Function barplotSignifSignatures in Package 'phenoTest'

Description

Methods for function `barplotSignifSignatures` in Package 'phenoTest'. For more information read the function's manual.

Methods

`signature(x = "epheno", signatures = "character")` Method for an `epheno` object and one signature stored in an object of class `character`.

`signature(x = "epheno", signatures = "list")` Method for an `epheno` object and several signatures stored in an object of class `list`.

`signature(x = "epheno", signatures = "GeneSet")` Method for an `epheno` object and one signature stored in an object of class `GeneSet`.

`signature(x = "epheno", signatures = "GeneSetCollection")` Method for an `epheno` object and several signatures stored in an object of class `GeneSetCollection`.

`epheno-class` *Class "epheno"*

Description

Object obtained with the ExpressionPhenoTest function. Contains FC, HR and pvals from testing expression values of each gene against phenotypic variables.

Objects from the Class

Objects can be created by calls of the form `new("epheno", assayData, phenoData, featureData, exprs, ...)`.

Slots

`p.adjust.method`**:** Object of class `"character"` containing the multiple testing adjustment method used (if one was used).

`assayData`**:** Object of class `"AssayData"` that is inherited from the `ExpressionSet` object used to create the `epheno` object.

`phenoData`**:** Object of class `"AnnotatedDataFrame"` that contains information about the variables stored in the experimentData slot such as their class (continuous, categorical, etc) or type (mean, summaryDif, pval, etc).

`featureData`: Object of class `"AnnotatedDataFrame"` that is inherited from the `ExpressionSet` object used to create the `epheno` object.

`experimentData`: Object of class `"MIAME"` that is inherited from the `ExpressionSet` object used to create the `epheno` object.

`annotation`: Object of class `"character"` that is inherited from the `ExpressionSet` object used to create the `epheno` object.

`protocolData`: Object of class `"AnnotatedDataFrame"` that is inherited from the `ExpressionSet` object used to create the `epheno` object.

`.__classVersion__`: Object of class `"Versions"` that is inherited from the `ExpressionSet` object used to create the `epheno` object.

Extends

Class `"ExpressionSet"`, directly. Class `"eSet"`, by class "ExpressionSet", distance 2. Class `"VersionedBiobase"`, by class "ExpressionSet", distance 3. Class `"Versioned"`, by class "ExpressionSet", distance 4.

Methods

[`signature(x = "epheno", i = "ANY", j = "ANY")`: inherited from the `ExpressionSet` class.

dim `signature(x = "epheno")`: inherited from the `ExpressionSet` class.

export2CSV `signature(x = "epheno")`: ...

getFc `signature(x = "epheno")`: getter for the fold changes.

getHr `signature(x = "epheno")`: getter for the hazard ratios.

getMeans `signature(x = "epheno")`: getter for the means.

getPvals `signature(x = "epheno")`: getter for the pvalues.

getSummaryDif `signature(x = "epheno")`: getter that returns hazard ratios, fold changes and pvalues.

gseaSignatures `signature(x = "epheno", signatures = "list")`: Used to compute GSEA. Please read the gseaSignatures manual.

logFcHr `signature(x = "epheno")`: getter for the log of fold changes and hazard ratios.

p.adjust.method `signature(x = "epheno")`: getter for the p value adjustment method that has been used.

phenoClass `signature(x = "epheno")`: Returns the class off all variables.

phenoNames `signature(x = "epheno")`: Returns the names of the tested phenotypes.

show `signature(object = "epheno")`: Shows a brief overview of the object.

Author(s)

Evarist Planet

Examples

```
showClass("epheno")
```

epheno *epheno object.*

Description

Object obatined with ExpressionPhenoTest function.

Usage

```
data(epheno)
```

Format

The format is: Formal class 'epheno' [package "phenoTest"] with 8 slots ..@ p.adjust.method : chr "none" ..@ assayData :<environment: 0x1050d5a78> ..@ phenoData :Formal class 'AnnotatedDataFrame' [package "Biobase"] with 4 slots .. .. ..@ varMetadata :'data.frame': 5 obs. of 1 variable: .. ..$ labelDescription: chr [1:5] NA NA NA NA ... .@ data :'data.frame': 12 obs. of 5 variables: .. ..$ phenoName : Factor w/ 3 levels "lymph.node.status",..: 3 3 3 3 3 1 1 1 2 3 ... .$ phenoClass: Factor w/ 3 levels "categorical",..: 2 2 2 2 2 1 1 1 3 2 ... .$ phenoType : Factor w/ 3 levels "mean","pval",..: 1 1 1 3 3 1 1 3 3 2 ... .$ meanLabel : Factor w/ 5 levels "[45.2,49.2)",..: 1 2 3 NA NA 4 5 NA NA NA ... .$ survTime : Factor w/ 1 level "Months2Relapse": NA NA NA NA NA NA NA NA 1 NA ... .@ dimLabels : chr [1:2] "sampleNames" "sampleColumns" .. .. ..@ .__classVersion__:Formal class 'Versions' [package "Biobase"] with 1 slots .. ..@ .Data:List of 1 .. ..$: int [1:3] 1 1 0 ..@ featureData :Formal class 'AnnotatedDataFrame' [package "Biobase"] with 4 slots .. .. ..@ varMetadata :'data.frame': 0 obs. of 1 variable: .. ..$ labelDescription: chr(0) .. .. ..@ data :'data.frame': 1000 obs. of 0 variables .. .. ..@ dimLabels : chr [1:2] "featureNames" "featureColumns" .. .. ..@ .__classVersion__:Formal class 'Versions' [package "Biobase"] with 1 slots .. ..@ .Data:List of 1 .. ..$: int [1:3] 1 1 0 ..@ experimentData :Formal class 'MIAME' [package "Biobase"] with 13 slots .. .. ..@ name : chr "" .. .. ..@ lab : chr "" .. .. ..@ contact : chr "" .. .. ..@ title : chr "" .. .. ..@ abstract : chr "" .. .. ..@ url : chr "" .. .. ..@ pubMedIds : chr "" .. .. ..@ samples : list() .. .. ..@ hybridizations : list() .. .. ..@ normControls : list() .. .. ..@ preprocessing : list() .. .. ..@ other : list() .. .. ..@ .__classVersion__:Formal class 'Versions' [package "Biobase"] with 1 slots .. ..@ .Data:List of 1 .. ..$: int [1:3] 1 0 0 ..@ annotation : chr "hgu133a" ..@ protocolData :Formal class 'AnnotatedDataFrame' [package "Biobase"] with 4 slots .. .. ..@ varMetadata :'data.frame': 0 obs. of 1 variable: .. ..$ labelDescription: chr(0) .. .. ..@ data :'data.frame': 12 obs. of 0 variables .. .. ..@ dimLabels : chr [1:2] "sampleNames" "sampleColumns" .. .. ..@ .__classVersion__:Formal class 'Versions' [package "Biobase"] with 1 slots .. ..@ .Data:List of 1 .. ..$: int [1:3] 1 1 0 ..@ .__classVersion__:Formal class 'Versions' [package "Biobase"] with 1 slots .. .. ..@ .Data:List of 4 .. ..$: int [1:3] 2 12 0 .. ..$: int [1:3] 2 10 0 .. ..$: int [1:3] 1 3 0 .. ..$: int [1:3] 1 0 0

Examples

```
data(epheno)
## maybe str(epheno) ; plot(epheno) ...
```

epheno2html *Create html files and plots from an epheno object.*

Description

Creates html files and plots using an epheno object, which stores the association between a list of variables and gene expression.

Usage

```
epheno2html(x, epheno, outputdir, prefix = "", genelimit = 50, categories = 3, w
```

Arguments

x	An object of class ExpressionSet (used to generate the epheno object) containing expression levels in exprs(x), phenotype information in pData(x) and annotation in annotation(x).
epheno	an object produced by ExpressionPhenoTest. this object will contain univariate association between a list of phenotype variables and gene expression as weel as p-values.
outputdir	where to place files.
prefix	will be used to add a text to the beginning of the files that will be created.
genelimit	maximum number of genes on the list.
categories	Number of categories used for continuous variables. It has to be the same as the one used for ExpressionPhenoTest.
withPlots	when FALSE no plots will be produced. Makes the process faster.
id.entrezid	this is for the particular case in which x's featureNames is not storing probeset ids but entrezids. The ExpressionSet will be storing information at the gene level and the row identidiers will be entrezids.
organism	this information will be used to query the BioMart database in order to get feature information. It will be used only if id.entrezid is TRUE.
homol.symbol	the homology table can be provided. This is useful if we want to run the same/similar code several times. This one will store the homology between entrezid and gene symbol. It will be used only if id.entrezid is TRUE. An example on how to create the homology table is shown in the examples section.
homol.genename	the same as homol.symbol but in this case the homology table stores the homology between entrezid and gene name.
mc.cores	number of cores that will be used to run the process.

Author(s)

Evarist Planet

Examples

```
#Example on building homology tables for human.
#mart <- useMart("ensembl", "hsapiens_gene_ensembl")
#homol.symbol <- getLDS(attributes = c("entrezgene"),
#    mart = mart, attributesL = c("external_gene_id"),
#    martL = mart, filters = "entrezgene", values = entrezid)
#mart <- useMart("ensembl", "hsapiens_gene_ensembl")
#homol.genename <- getLDS(attributes = c("entrezgene"),
#    mart = mart, attributesL = c("description"), martL = mart,
#    filters = "entrezgene", values = entrezid)
```

---

getters for the epheno object

Getters for the epheno object:

Description

`getFc` gets the fold changes. `getHr` gets the hazard ratios. `getMeans` gets the means. `getPvals` gets the p values. `getSummaryDif` gets fold changes and hazard ratios. `lofFcHr` gets the fold changes and hazard ratios after log scaling. `p.adjust.method` gets the p value adjustment method that was used when creating the object. `phenoClass` returns a data.frame telling the class (ordinal, continuous, categorical or survival) of each phenotype. `phenoNames` gets the phenotype names.

Usage

```
getFc(x)
getHr(x)
getMeans(x)
getPvals(x)
getSummaryDif(x)
logFcHr(x)
p.adjust.method(x)
phenoClass(x)
phenoNames(x)
```

Arguments

`x`	`epheno` object

Author(s)

Evarist Planet

`eset` *Example data.*

Description

Example data of class ExpressionSet.

Usage

```
data(eset)
```

Format

The format is: Formal class 'ExpressionSet' [package "Biobase"] with 7 slots ..@ assayData :<environment: 0x1050d9390> ..@ phenoData :Formal class 'AnnotatedDataFrame' [package "Biobase"] with 4 slots .. .. ..@ varMetadata :'data.frame': 7 obs. of 1 variable: .. ..$ labelDescription: chr [1:7] NA NA NA NA ... .@ data :'data.frame': 286 obs. of 7 variables: .. ..$ PID : int [1:286] 3 5 6 7 8 9 11 14 15 17 ... .$ GEOaccession : Factor w/ 286 levels "GSM36777","GSM36778",..: 17 20 21 22 24 25 58 59 60 61 ... .$ lymph.node.status: chr [1:286] "negative" "negative" "negative" "negative" ... .$ Months2Relapse : int [1:286] 101 118 9 106 37 125 109 14 99 137 ... .$ Relapse : int [1:286] 0 0 1 0 1 0 0 1 0 0 ... .$ ER.Status : num [1:286] 0 1 0 0 0 1 1 0 1 1 ... .$ BrainRelapse : int [1:286] 0 0 0 0 0 0 0 0 0 0 ... .@ dimLabels : chr [1:2] "sampleNames" "sampleColumns" .. .. ..@ .__classVersion__:Formal class 'Versions' [package "Biobase"] with 1 slots .. ..@ .Data:List of 1 .. ..$: int [1:3] 1 1 0 ..@ featureData :Formal class 'AnnotatedDataFrame' [package "Biobase"] with 4 slots .. .. ..@ varMetadata :'data.frame': 16 obs. of 3 variables: .. ..$ Column : chr [1:16] "ID" "GB_ACC" "SPOT_ID" "Species Scientific Name" ... .$ Description : Factor w/ 15 levels "","A gene symbol, when one is available (from UniGene).",..: 3 5 15 13 12 1 11 1 10 14 ... .$ labelDescription: chr [1:16] NA NA NA NA ... .@ data :'data.frame': 1000 obs. of 16 variables: .. ..$ ID : Factor w/ 22284 levels "1007_s_at","1053_at",..: 1 2 3 4 5 6 7 8 9 10 ... .$ GB_ACC : Factor w/ 21129 levels "AF052179","AF061832",..: 93 30 95 97 25 24 96 99 28 20 ... .$ SPOT_ID : chr [1:1000] NA NA NA NA ... .$ Species.Scientific.Name : Factor w/ 2 levels "Homo sapiens",..: 1 1 1 1 1 1 1 1 1 1 ... .$ Annotation.Date : Factor w/ 2 levels "Jul 11, 2007",..: 1 1 1 1 1 1 1 1 1 1 ... .$ Sequence.Type : Factor w/ 4 levels "Consensus sequence",..: 2 2 2 2 2 2 2 2 2 2 ... .$ Sequence.Source : Factor w/ 3 levels "Affymetrix Proprietary Database",..: 1 2 1 2 1 2 1 1 2 1 ... .$ Target.Description : Factor w/ 21363 levels "Consensus includes gb:AI656011 /FEA=EST /DB_XREF=gi:4739990 /DB_XREF=est:tt42e08.x1 /CLONE=IMAGE:2243462 /UG=Hs.116875 KIAA0156"| __truncated__,..: 16 13 18 20 8 7 19 22 11 4 ... .$ Representative.Public.ID : Factor w/ 21197 levels "AF052179","AF061832",..: 93 30 95 97 25 24 96 99 28 20 ... .$ Gene.Title : Factor w/ 14208 levels "ADP-ribosylation factor 1",..: 35 66 46 60 44 97 96 64 26 33 ... .$ Gene.Symbol : Factor w/ 13293 levels "ABCF1","ARF1",..: 20 59 40 53 33 96 94 58 15 18 ... .$ ENTREZ_GENE_ID : chr [1:1000] "780" "5982" "3310" "7849" ... .$ RefSeq.Transcript.ID : Factor w/ 13074 levels "NM_000409","NM_000661 /// NM_001024921",..: 37 45 41 52 1 50 49 82 47 4 ... .$ Gene.Ontology.Biological.Process: Factor w/ 7245 levels "","0000074 // regulation of progression through cell cycle // traceable author statement /// 0006139 // nucleobase, nucleoside, nu"| __truncated__,..: 61 22 78 32 79 60 14 63 72 20 ... .$ Gene.Ontology.Cellular.Component: Factor w/ 4148 levels "","0000502 // proteasome complex (sensu Eukaryota) // traceable author statement /// 0005634 // nucleus // inferred from electroni"| __truncated__,..: 72 45 1 44 1 1 42 71 6 68 ... .$ Gene.Ontology.Molecular.Function: Factor w/ 7314 levels "","0000049 // tRNA binding // non-traceable author statement /// 0000166 //

nucleotide binding // inferred from electronic annotat"| __truncated__,..: 23 26 27 40 81 18 39 71 74 69 ... .@ dimLabels : chr [1:2] "featureNames" "featureColumns" .. .. ..@ .__classVersion__:Formal class 'Versions' [package "Biobase"] with 1 slots .. ..@ .Data:List of 1 .. ..$: int [1:3] 1 1 0 ..@ experimentData :Formal class 'MIAME' [package "Biobase"] with 13 slots .. .. ..@ name : chr "" .. .. ..@ lab : chr "" .. .. ..@ contact : chr "" .. .. ..@ title : chr "" .. .. ..@ abstract : chr "" .. .. ..@ url : chr "" .. .. ..@ pubMedIds : chr "" .. .. ..@ samples : list() .. .. ..@ hybridizations : list() .. .. ..@ normControls : list() .. .. ..@ preprocessing : list() .. .. ..@ other : list() .. .. ..@ .__classVersion__:Formal class 'Versions' [package "Biobase"] with 1 slots .. ..@ .Data:List of 1 .. ..$: int [1:3] 1 0 0 ..@ annotation : chr "hgu133a" ..@ protocolData :Formal class 'AnnotatedDataFrame' [package "Biobase"] with 4 slots .. .. ..@ varMetadata :'data.frame': 0 obs. of 1 variable: .. ..$ labelDescription: chr(0) .. .. ..@ data :'data.frame': 286 obs. of 0 variables .. .. ..@ dimLabels : chr [1:2] "sampleNames" "sampleColumns" .. .. ..@ .__classVersion__:Formal class 'Versions' [package "Biobase"] with 1 slots .. ..@ .Data:List of 1 .. ..$: int [1:3] 1 1 0 ..@ .__classVersion__:Formal class 'Versions' [package "Biobase"] with 1 slots .. .. ..@ .Data:List of 4 .. ..$: int [1:3] 2 12 0 .. ..$: int [1:3] 2 10 0 .. ..$: int [1:3] 1 3 0 .. ..$: int [1:3] 1 0 0

References

Has been obtained from GEO (GSE2034). Only first 1000 genes where stored (the rest has been removed).

Examples

```
data(eset)
## maybe str(eset) ; plot(eset) ...
```

---

export2CSV-methods *Methods for Function export2CSV in Package 'phenoTest'*

Description

Methods for function `export2CSV` in Package 'phenoTest'

Methods

`signature(x = "epheno")` Exports summary differences (fold changes, hazard ratios), p-values and gene annotation (when available) to a CSV (comma separated value) file

---

export2CSV *Export object to comma-separated text file.*

Description

Saves object as comma-separated text file (CSV), using `write.csv`.

Usage

```
export2CSV(x, file, row.names=FALSE, ...)
```

Arguments

x	object to be exported. Currently methods for objects of class epheno (produced with ExpressionPhenoTest function) are implemented.
file	Name of the file where the results are to be saved
row.names	Passed on to write.csv
...	Other arguments to be passed on to write.csv

---

findCopyNumber *Find copy number regions using expression data in a similar way ACE does.*

Description

Given enrichment scores between two groups of samples and the chromosomical positions of those enrichment scores this function finds areas where the enrichemnt is bigger/lower than expected if the positions where assigned at random. Plots of the regions and positions of the enriched regions are provided.

Usage

```
findCopyNumber(x, minGenes = 15, B = 100, p.adjust.method = "BH",
pvalcutoff = 0.05, exprScorecutoff = NA, mc.cores = 1, useAllPerm = F,
genome = "hg19", chrLengths, sampleGenome = TRUE, useOneChr = FALSE,
useIntegrate = TRUE,plot=TRUE)
```

Arguments

x	An object of class data.frame with gene or probe identifiers as row names and the following columns: es (the enrichment score), chr (the chromosome where the gene or probe belong to) and pos (position in the chromosome in megabases). It can be obtained (from an epheno object) with the function getEsPositions.
minGenes	Minimum number of genes in a row that have to be enriched to mark the region as enriched. Has to be bigger than 2.
B	Number of permuations that will be computed to calculate pvalues. If useAllPerm is FALSE this value has to be bigger than 100. If useAllPerm is TRUE the computations are much more expensive, therefore it is not recommended to use a B bigger than 100.
p.adjust.method	P value adjustment method to be used. p.adjust.methods provides a list of available methods.
pvalcutoff	All genes with an adjusted p value lower than this parameter will be considered enriched.
exprScorecutoff	Genes with a smoothed score that is not bigger (lower if the given number is negative) than the specified value will not be considered significant.
mc.cores	Number of cores to be used in the computation. If mc.cores is bigger than 1 the multicore library has to be loaded.

useAllPerm	If FALSE for each gene only permutations of genes that are in an area with similar density (similar number of genes close to them) are used to compute pvalues. If TRUE all permutations are used for each gene. We recommend to use the option FALSE after having observed that the enrichment can depend on the number of genes that are in the area. We recommend to use the option TRUE if the positions of the enrichment score are equidistant. Take into account that this option is much slower and needs less permutations, therefore a smaller B is preferred. See details for more info.
genome	Genome that will be used to draw cytobands.
chrLengths	An object of class numeric containing chromosome names as names. This names have to be the same as the ones used in x$chr If missing the last position is used.
sampleGenome	If positions are sampled over the hole genome (across chromosomes) or within each chromosome. This is TRUE by default.
useOneChr	Use only one chromosome to build the distribution under the null hypothesis that genes/probes are not enriched. By default this is FALSE. The chromosome that is used is chosen as follows: after removing small chromosomes we select the one closest to the median quadratic distance to 0. Setting this parameter to TRUE decreases processing time.
useIntegrate	If we want to use integrate or pnorm to compute pvalues. The first does not assume any distribution for the distribution under the null hypothesis, the second assumes it is normally distributed.
plot	If FALSE the function will make no plots.

Details

Enrichemnt scores can be either log fold changes, log hazard ratios, log variabiliy ratios or any other score.

Within each chromosome a smoothed score for each gene is obtained via generalized additive models, the smoothing parameter for each chromosome being chosen via cross-validation. The obtained smoothing parameter of each chromosome will be used in permutations.

We assessed statistical significance by permuting the positions thrue the hole genome. If useAllPerm is FALSE for each gene the permutations of genes that are in an area with similar density (distance to tenth gene) are used to compute pvalues. We observed that genes with similar densities tend to have similar smoothed scores. If we set 1000 permutations (B=1000) scores are permuted thrue the hole genome 10 times (1000/100). For each smoothed scored the permutations of the 100 smoothed scores with most similar density (distance to tenth gene) are used. Therefore each smoothed score will be compared to 1000 smoothed scores obtained from permutations.

If scores are at the same distance in the genome from each other (for instance when we have a score every fixed certain bases) the option useAllPerm=TRUE is recommended. In this case every smoothed score is compared to all smoothed scores obtained via permutations. In this case having 20,000 genes and setting the paramter B=10 would mean that the scores are permuted 10 times times thrue the hole genome, obtaining 200,000 permuted smoothed scores. Each observed smoothed score will be tested against the distribution of the 200,000 permuted smoothed scores.

Only regions with as many genes as told in minGenes being statistically significant (pvalue lower than parameter pvalcutoff) after adjusting pvalues with the method specified in p.adjust.method will be selected as enriched. If exprScorecutoff is different form NA, a gene to be statistically significant will need (aditionally to the pvalue cutoff) to have a smoothed score bigger (lower if exprScorecutoff is negative) than the specified value.

Value

Plots all chromomes and marks the enriched regions. Also returns a `data.frame` containing the positions of the enriched regions. This output can be passed by to the `genesInArea` function to obtain the names of the genes that are in each region.

Author(s)

Evarist Planet

See Also

getEsPositions, genesInArea

Examples

```
#data(epheno)
#phenoNames(epheno)
#mypos <- getEsPositions(epheno,'Relapse')
#mypos$chr <- '1' #we set all probes to chr one for illustration purposes
#                 #(we want a minimum number of probes per chromosome)
#head(mypos)
#set.seed(1)
#regions <- findCopyNumber(mypos,B=10,plot=FALSE)
#head(regions)
```

`genesInArea` *Find genes that are in given areas.*

Description

Combine the output of getEsPositions and findCopyNumber to see which genes are in the enriched areas.

Given areas of enrichment (obtained with `findCopyNumber`) and a set of genes or probes and their positions in the genome (obtained with `getEsPositions`) the function tells which genes fall in each area.

Usage

```
genesInArea(x, regions)
```

Arguments

`x`	An object of class `data.frame` with gene or probe identifiers as row names and the following columns: es (the enrichment score), chr (the chromosome where the gene or probe belong to) and pos (position in the chromosome in megabases). It can be obtained with the function `getEsPositions`.
`regions`	This is usually the output of `findCopyNumber` function.

Author(s)

Evarist Planet

See Also

getEsPositions, findCopyNumber

Examples

```
data(epheno)
phenoNames(epheno)
mypos <- getEsPositions(epheno,'Relapse')
head(mypos)
#regions <- findCopyNumber(mypos)
#head(regions)
#genes <- genesInArea(mypos,regions)
#head(genes)
```

`getEsPositions` *Obtain chromosome positions for each gene.*

Description

Given an object of class epheno obtain the gene positions on the genome.

Usage

```
getEsPositions(epheno, phenoName, organism = "human", logEs = T, center = FALSE)
```

Arguments

`epheno`	An object of class `epheno` usually obtained with `ExpressionPhenoTest`
`phenoName`	The phenotype that we want to use. Has to be in `phenoNames(epheno)`
`organism`	Has to be 'human' or 'mouse'. The default is 'human'.
`logEs`	If the values have to be log scaled.
`center`	If the values have to be genome centered. If TRUE the genome average will be substracted to every value.

Details

The output will usually be passed to findCopyNumber.

Value

An object of class `data.frame` will be returned containing 3 variables: es (enichment score for fold change or hazard ratio), chr (chromosome), pos (position in Mb). epheno's featureNames will be used as row names.

Author(s)

Evarist Planet

Examples

```
data(epheno)
phenoNames(epheno)
mypos <- getEsPositions(epheno,'Relapse')
head(mypos)
```

---

get gseaSignatures' elements

Substract element's of a gseaSignaturesSign or gseaSignaturesVar object (obtained using the gseaSignatures function).

Description

getEs returns ES (enrichment scores) getEsSim returns simulated ES (needed to compute pvals), getNes returns NES (normalized enrichment scores) and getFcHr returns the fold changes or hazard used to compute the ES, simulated ES and NES.

Usage

```
getEs(x)
getEsSim(x)
getNes(x)
getFcHr(x)
```

Arguments

x	an gseaSignaturesSign or gseaSignaturesVar object. Those objects are obtained using the gseaSignatures function.

Author(s)

Evarist Planet

---

getVars2test-methods

Methods for Function getVars2test in Package 'phenoTest'

Description

Methods for function getVars2test in Package 'phenoTest'. For more information read the funcion's manual.

Methods

signature(x = "epheno") Method for an object of class epheno.

getVars2test *Get phenotypic variables that were tested.*

Description

Returns an object containing the names of the variables that were tested when the epheno object was created. Will return an object of class list. Variables of the same type (categorical, survival, etc) will be in the same slot of the list. The slot names are the types of the variables.

Author(s)

Evarist Planet

Examples

```
data(epheno)
getVars2test(epheno)
```

gseaSignatures-class *Class "gseaSignatures" ES and EsSim container.*

Description

This object contains de ES (enrichment scores) and simulated ES that will be used in the GSEA (Gene Set Enrichment Analysis) process.

Objects from the Class

Objects can be created by calls of the form `new("gseaSignatures", ...)`.

Slots

`.Data`: Object of class `"list"` .

`es`: Object of class `"numeric"` Contains the observed enrichment scores. The ones that were compted from the data without permuting anything.

`es.sim`: Object of class `"numeric"` Contains the enrichment score that were obtained after permutations.

`signature`: Object of class `"numeric"` The subset of genes we are interested in.

Extends

Class `"list"`, from data part. Class `"vector"`, by class "list", distance 2. Class `"AssayData"`, by class "list", distance 2.

Methods

No methods defined with class "gseaSignatures" in the signature.

Author(s)

Evarist Planet

Examples

```
showClass("gseaSignatures")
```

---

gseaSignatures-methods *Methods for Function gseaSignatures in Package 'phenoTest'*

Description

Methods for function `gseaSignatures` in Package 'phenoTest'. For more information read the function's manual.

Methods

`signature(x = "ANY", signatures = "character")` Method for signature of class `character`.

`signature(x = "ANY", signatures = "GeneSet")` Method for signature of class `character`.

`signature(x = "epheno", signatures = "list")` Method for an `epheno` object and several signatures stored in an object of class `list`.

`signature(x = "matrix", signatures = "GeneSetCollection")` Method for an `matrix` object and several signatures stored in an object of class `GeneSetCollection`.

`signature(x = "epheno", signatures = "GeneSetCollection")` Method for an `epheno` object and several signatures stored in an object of class `GeneSetCollection`.

`signature(x = "numeric", signatures = "GeneSetCollection")` Method for an `numeric` object and several signatures stored in an object of class `GeneSetCollection`.

`signature(x = "matrix", signatures = "list")` Method for an `matrix` object and several signatures stored in an object of class `list`.

`signature(x = "numeric", signatures = "list")` Method for an `numeric` object and several signatures stored in an object of class `list`.

---

gseaSignatures *Compute ES (enrichment scores) and es.sim (simulated enrichment scores) for given phenotipic variable(s) and signature(s).*

Description

This function computes the first step in the process of obtaining a GSEA-like plot. It computes the enrichment scores and simulated enrichment scores for each variable and signature. The output will usually be used as input for the `gseaSignificance` function. An important parameter of the function is `logScale`. Its default value is TRUE which means that by default the provided scores (i.e. fold changes, hazard ratios) will be log scaled. Remember to change this parameter to FALSE if your scores are already log scaled. The `getEs`, `getEsSim`, `getFc`, `getHr` and `getFcHr` methods can be used to acces each subobject. For more information please visit the man pages of each method.

Usage

```
gseaSignatures(x,signatures,logScale=TRUE,absVals=FALSE,averageRepeats=FALSE,B=1
```

Arguments

x	ePhenoTest, numeric or matrix object containing hazard ratios or fold changes.
signatures	character or list object containing the names of the genes that belong to each signature.
logScale	if values should be log scaled.
absVals	if TRUE fold changes and hazard ratios that are negative will be turned into positive before starting the process. This is useful when genes can go in both directions.
averageRepeats	if x is of class numeric and has repeated names (several measures for some indivdual names) we can average the measures of the same names.
B	number of simulations to perform.
mc.cores	number of processors to use.
test	the test that will be used. 'perm' stands for the permutation based method, 'wilcox' stands for the wilcoxon test (this is the fastest one) and 'ttperm' stands for permutation t test.

Details

The following preprocessing was done on the provided scores (i.e. fold changes, hazard ratios) to avoid errors during the enrichment score computation: -When having two scores with the same name its average was used. -Zeros were removed. -Scores without names (which can not be in any signature) removed. -Non complete cases (i.e. NAs, NaNs) were removed. ES score was calculated for each signature and variable (see references). If parameter test is 'perm' the signature was permutted and the ES score was recalculated (this happened B times for each variable, 1000 by default). If test is 'wilcox' a wilcoxon test in which we test the fact that the average value of the genes that do belong to our signtaure is different from the average value of the genes that do not belong to our signature will be performed. If test is 'ttperm' a permutation t-test will be used. Take into account that the final plot will be different when 'wilcox' is used.

Author(s)

Evarist Planet

References

Aravind Subramanian, (October 25, 2005) *Gene Set Enrichment Analysis*. www.pnas.org/cgi/doi/10.1073/pnas.0506580102

Examples

```
#load epheno object
data(epheno)
epheno

#we construct two signatures
sign1 <- sample(featureNames(epheno))[1:20]
```

```
sign2 <- sample(featureNames(epheno))[50:75]
mySignature <- list(sign1,sign2)
names(mySignature) <- c('My first signature','My preferred signature')

#run gsea functions
#my.gseaSignatures <- gseaSignatures(x=epheno,signatures=mySignature,B=100,mc.cores=1)
#my.gseaSignificance <- gseaSignificance(my.gseaSignatures)
#my.summary <- summary(my.gseaSignificance)
#my.summary
#plot(my.gseaSignatures,my.gseaSignificance)
```

`gseaSignaturesSign-class` *Class "gseaSignaturesSign"*

Description

This class is an ES (enrichment score) and ES.sim (simulated enrichment score) container that will be used in the GSEA (Gene Set Enrichment Analysis) process. There is one container for every gene signature.

Objects from the Class

Objects can be created by calls of the form `new("gseaSignaturesSign", ...)`.

Slots

`.Data`: Object of class `"list"`.

`gseaSignatures`: Object of class `"gseaSignatures"` This is the object that will contain the ES and ES.sim.

`fc.hr`: Object of class `"character"` fold change or hazard ratio used to compute the enrichment scores.

`s`: Object of class `"logical"` The subset of genes we are interested in.

`test`: Object of class `"character"` The statistical test that will be used.

Extends

Class `"list"`, from data part. Class `"vector"`, by class "list", distance 2. Class `"AssayData"`, by class "list", distance 2.

Methods

getEs `signature(x = "gseaSignaturesSign")`: Returns the enrichment scores.

getEsSim `signature(x = "gseaSignaturesSign")`: Returns the simulated enrichemt scores (the ones obtained after permutations).

getFcHr `signature(x = "gseaSignaturesSign")`: Returns the fold change and/or the hazard ratio that were used to compute the enrichment scores.

gseaSignificance `signature(x = "gseaSignaturesSign")`: This is the next step in the process of performing GSEA. This funtion will test if the gene sets are enriched.

Author(s)

Evarist Planet

Examples

```
showClass("gseaSignaturesSign")
```

---

gseaSignaturesVar-class

Class "gseaSignaturesVar"

Description

This class is an ES (enrichment score) and ES.sim (simulated enrichment score) container that will be used in the GSEA (Gene Set Enrichment Analysis) process. There is one container for every phenotype. Every one of this containers (of class `gseaSignaturesSign`) is a container itsef and has the enrichment scores of all signatures. `GseaSignaturesVar` contains one element per phenotype (phenotypic variable). Every one of this elements is of class `gseaSignaturesSign` and contains one element per signature.

Objects from the Class

Objects can be created by calls of the form `new("gseaSignaturesVar", ...)`.

Slots

`.Data`**:** Object of class `"list"`.

`gseaSignatures`**:** Object of class `"gseaSignaturesSign"` This object contains the enrichment scores and other elements that will be used in the GSEA process.

Extends

Class `"list"`, from data part. Class `"vector"`, by class "list", distance 2. Class `"AssayData"`, by class "list", distance 2.

Methods

getEs `signature(x = "gseaSignaturesVar")`: Returns the enrichment scores.

getEsSim `signature(x = "gseaSignaturesVar")`: Returns the simulated enrichemt scores (the ones obtained after permutations).

getFcHr `signature(x = "gseaSignaturesVar")`: Returns the fold change and/or the hazard ratio that were used to compute the enrichment scores.

gseaSignificance `signature(x = "gseaSignaturesVar")`: This is the next step in the process of performing GSEA. This funtion will test if the gene sets are enriched.

Author(s)

Evarist Planet

Examples

```
showClass("gseaSignaturesVar")
```

gseaSignificance *ES' (enrichment scores) sifgnificance.*

Description

This function performs the second step in the process of obtaining a GSEA-like plot. It computes the NES (normalized enrichment score), p values and fdr (false discovery rate) for all variables and signatures. A gseaSignaturesSign or gseaSignaturesVar object will be needed as input (these objects can be obtained with the gseaSignatures function). For an overview of the output use the summary method. The next step after using the gseaSignificance function would be using the plot method.

Usage

```
gseaSignificance(x,p.adjust.method='none',pval.comp.method='original',pval.smoot
```

Arguments

x	gseaSignaturesSign or gseaSignaturesVar object obtained with the gseaSignatures method. This object contains the enrichment scores ,the simulated enrichment scores and the fold changes or hazard ratios.
p.adjust.method	p adjustment method to be used. Common options are 'BH', 'BY', 'bonferroni' or 'none'. All available options and their explanations can be found on the p.adjust function manual.
pval.comp.method	the p value computation method. Has to be one of 'signed' or 'original'. The default one is 'original'. See details for more information.
pval.smooth.tail	if we want to estimate the tail of the ditribution where the pvalues will be generated.

Details

The simulated enrichment scores and the calculated one are used to find the p value.

P value calculation depends on the parameter pval.comp.method. The default value is 'original'. In 'original' we are simply computing the proportion of anbolute simulated ES which are larger than the observed absolute ES. In 'signed' we are computing the proportion of simulated ES which are larger than the observed ES (in case of having positive enrichment score) and the proportion of simulated ES which are smaller than the observed ES (in case of having negative enrichment score).

Author(s)

Evarist Planet

References

Aravind Subramanian, (October 25, 2005) *Gene Set Enrichment Analysis*. www.pnas.org/cgi/doi/10.1073/pnas.0506580102

C.A. Tsai and J.J. Chen. *Kernel estimation for adjusted p-values in multiple testing. Computational Statistics & Data Analysis* http://econpapers.repec.org/article/eeecsdana/v_3a51_3ay_3a2007_3ai_3a8_3ap_3a3885-3897.htm

Examples

```
#for examples see the help file of gseaSigntaures: ?gseaSignatures
```

---

gseaSignificanceSign-class

Class "gseaSignificanceSign"

Description

This object contains the results of the test of enrichment that was performed on each gene set. There is one container for every gene signature.

Objects from the Class

Objects can be created by calls of the form `new("gseaSignificanceSign", ...)`.

Slots

`.Data`: Object of class `"list"` .

`gseaSignificance`: Object of class `"matrix"` Contains the statistics. Use the `summary` method to access this information.

`p.adjust.method`: Object of class `"character"` The p-value adjustment method that was used.

Extends

Class `"list"`, from data part. Class `"vector"`, by class "list", distance 2. Class `"AssayData"`, by class "list", distance 2.

Methods

No methods defined with class "gseaSignificanceSign" in the signature.

Author(s)

Evarist Planet

Examples

```
showClass("gseaSignificanceSign")
```

gseaSignificanceVar-class

Class "gseaSignificanceVar"

Description

This object contains the results of the test of enrichment that was performed on each gene set and phenotype. There is one container for every phenotype. Every one of this containers (of class `gseaSignificanceSign`) is a container itsef and has the results of the tests for all signatures. `GseaSignificanceVar` contains one element per phenotype (phenotypic variable). Every one of this elements is of class `gseaSignificanceSign` and contains one element per signature.

Objects from the Class

Objects can be created by calls of the form `new("gseaSignificanceVar", ...)`.

Slots

`.Data`: Object of class `"list"` .

`gseaSignificance`: Object of class `"gseaSignificanceSign"` This object contains the results of the tests.

Extends

Class `"list"`, from data part. Class `"vector"`, by class "list", distance 2. Class `"AssayData"`, by class "list", distance 2.

Methods

No methods defined with class "gseaSignificanceVar" in the signature.

Author(s)

Evarist Planet

Examples

```
showClass("gseaSignificanceVar")
```

heatmapPhenoTest-methods

Methods for Function heatmapPhenoTest in Package 'phenoTest'

Description

Methods for function `heatmapPhenoTest` in Package 'phenoTest'. For more information read the function's manual.

Methods

`signature(x = "ExpressionSet", signatures = "character")` Method for an `ExpressionSet` object and one signature stored in an object of class `character`.

`signature(x = "ExpressionSet", signatures = "list")` Method for an `ExpressionSet` object and several signatures stored in an object of class `list`.

`signature(x = "ExpressionSet", signatures = "missing")` Method for an `ExpressionSet` object and no signatures.

`signature(x = "ExpressionSet", signatures = "GeneSet")` Method for an `ExpressionSet` object and one signature stored in an object of class `GeneSet`.

`signature(x = "ExpressionSet", signatures = "GeneSetCollection")` Method for an `ExpressionSet` object and several signatures stored in an object of class `GeneSetCollection`.

---

heatmapPhenoTest *Produce heatmap from phenotype data.*

Description

Show the associations between clusters that each sample belongs to and each phenotype in a heatmap and/or a Kaplan-Meier plot.

Usage

```
heatmapPhenoTest(x, signatures, vars2test, probes2genes = FALSE,
filterVar, filteralpha = 0.05, distCol = "pearson", nClust = 2, distRow
= "cor", p.adjust.method = "none", simulate.p.value = FALSE, B = 10^5,
linkage = "average", equalize = FALSE, center = TRUE, col, survCol,
heat.kaplan="both", ...)
```

Arguments

`x`	ExpressionSet with phenotype information stored in `pData(x)`.
`signatures`	Either character vector or list of character vectors with gene sets to be used to draw heatmaps (gene names should match those in `featureNames(x)`). A separate heatmap will be produced for each element in the list.
`vars2test`	list with components 'continuous', 'categorical', 'ordinal' and 'survival' indicating which phenotype variables should be tested. 'continuous', 'categorical' and 'ordinal' must be character vectors, 'survival' a matrix with columns named 'time' and 'event'. The names must match names in `names(pData(x))`.
`probes2genes`	If set to `TRUE` a single probe is selected for each gene. `nsFilter` is used to select the probe with highest inter-quartile range.
`filterVar`	If specified, only genes with significant differences in the variable `filterVar` will be displayed in the heatmap. Note that this option will not affec the sample clustering, as this is obtained using both significant and non-significant genes.
`filteralpha`	Significance level for the filtering based on `filterVar`.
`distCol`	Distance metric used to cluster columns (e.g. patients/samples). Can take any value accepted by `dist`. Pearson and Spearman correlations are also allowed. Write 'spearman' or 'pearson' to use them.

nClust	Number of desired clusters.
distRow	Distance metric used to cluster rows (e.g. genes). Can take any value accepted by distancematrix.
p.adjust.method	Method for P-value adjustment, passed on to p.adjust.
simulate.p.value	If set to FALSE the chi-square test p-values are computed using asymptotics, otherwise a simulation is used (see chisq.test for details).
B	An integer specifying the number of replicates used in the chi-square Monte Carlo test (passed on to chisq.test).
linkage	Linkage used for clustering. Must be either 'complete', 'average' or 'minimum'.
equalize	Should color codes be equalized between genes, i.e. all genes present the same range of colors. Passed on to heatmap_plus.
center	centering is done by subtracting the column means (omitting NAs).
col	Color scheme to be used for heatmap. Defaults to a green/red scheme designed to look nice for microarray data.
survCol	Colors for the Kaplan-Meier survival curves.
heat.kaplan	can be "heat" if we want to plot a heatmap, "kaplan" if we want to plot a kaplan-meier or "both" if we want both of them.
...	Other arguments for the survival plot, e.g. lty etc.

Details

Makes two clusters of samples based on the expression levels of the genes from the given signature and plots a heatmap and/or a Kaplan-Meier showing the association between belonging to one cluster or the other and each phenotype.

For variables in vars2test\$continuous and vars2test\$ordinal a Kruskal-Wallis Rank Sum test is used; for vars2test\$categorical a chi-square test (with exact p-value if simulate.p.value is set to TRUE); for var2test\$survival a Cox proportional hazards likelihood-ratio test.

Author(s)

David Rossell

Examples

```
#load data
data(eset)
eset

#construct vars2test
survival <- matrix(c("Relapse","Months2Relapse"),ncol=2,byrow=TRUE)
colnames(survival) <- c('event','time')
vars2test <- list(survival=survival)
vars2test

#construct a signature
sign <- sample(featureNames(eset))[1:20]

#make plot
heatmapPhenoTest(eset,sign,vars2test=vars2test,heat.kaplan='heat')
heatmapPhenoTest(eset,sign,vars2test=vars2test,heat.kaplan='kaplan')
```

htmlpage *htmlpage*

Description

This is a copy of the `htmlpage` function from annotate `package`. The copy was made to allow htmlpage to produce links to local files.

Details

See `?annotate::htmlpage` fror more information.

pAdjust-methods *Methods for Function pAdjust in Package 'phenoTest'*

Description

Methods for function `pAdjust` in Package 'phenoTest'. This function adjusts the p-values of an `epheno` object. For more informe read the function's manual.

Methods

`signature(x = "epheno")` Adjusts the pvalues of an epheno object.

Author(s)

Evarist Planet

pAdjust *Adjust p values of an epheno object.*

Description

Adjusts the p values of an epheno object. The `p.adjust` function will be used. For more information read the `p.adjust` function's help.

Usage

```
pAdjust(x, method = "BH")
```

Arguments

`x`	an `epheno` object.
`method`	the correction method that will be used. Se the `p.adjust` help for more info about the methods.

Author(s)

Evarist Planet

Examples

```
#load epheno object
data(epheno)
epheno

#Adjust pvalue
p.adjust.method(epheno)
epheno <- pAdjust(epheno,method='BH')
p.adjust.method(epheno)
```

---

phenoTest-package *Test correlation between gene expression and phenotype.*

Description

Test correlation between gene expression and phenotype.

Details

Package:	phenoTest
Type:	Package
Version:	1.0
Date:	2010-04-28
License:	What license is it under?
LazyLoad:	yes

Author(s)

Evarist Planet Maintainer: Evarist Planet <evarist.planet@irbbarcelona.org>

---

plot.gseaSignatures *GSEA-like Plot.*

Description

Builds a GSEA plot using a gseaSignature object (one of `gseaSignaturesSign` or `gseaSignaturesVar` obtained with the `gseaSignatures` function) and a gseaSignificance object (one of `gseaSignificanceSign` or `gseaSignificanceVar` obtained with the `gseaSignificance` function).

Usage

```
plot.gseaSignaturesSign(x,gseaSignificance,es.ylim,nes.ylim,es.nes="both",...)
plot.gseaSignaturesVar(x,gseaSignificance,es.ylim,nes.ylim,es.nes="both",...)
```

Arguments

x	object of class gseaSignaturesSign or gseaSignatures Var.
gseaSignificance	object of class gseaSignificanceSign or gseaSignificanceVar.
es.ylim	ylim values for the ES plot.
nes.ylim	ylim values for the NES plot.
es.nes	can be "es" if we want to plot enrichment score, "nes" if we want to plot normalised enrichment scores or "both"if we want to plot them both.
...	Arguments to be passed to plot.

Author(s)

Evarist Planet

References

Aravind Subramanian, (October 25, 2005) *Gene Set Enrichment Analysis*. www.pnas.org/cgi/doi/10.1073/pnas.0506580102

Examples

```
#for examples see the help file of gseaSigntaures: ?gseaSignatures
```

---

barplotSignatures *Summary plots for gene signature vs phenotype association*

Description

Summarizes the univariate relationships between genes in one or more signatures and several phenotype variables, as summarized in epheno objects (which can be created with the ExpressionPhenoTest function).

By default barplotSignifSignatures performs a binomial test (binom.test from package stats) for each signature to see if the number of up up regulated and down regulated genes is different enough to be statistically different. When a reference gene set is provided we test if the proportions of up and down regulated genes of each gene set is different from the proportions in the reference gene set. This has been done with a chi-square test. When a reference gene set is provided and parameter testUpDown is TRUE (by default its FALSE) the number of genes corresponding to up and down regulated are compared with those of the reference gene set separately.

Usage

```
barplotSignatures(x, signatures, referenceSignature, alpha=.05,
p.adjust.method='none', ylab, cex.text=1, ...)
barplotSignifSignatures(x, signatures, referenceSignature, testUpDown=FALSE,
simulate.p.value = FALSE, B = 10^4, p.adjust.method='none', alpha=.05,
ylab, ylim=ylim, cex.text=1, ...)
```

Arguments

x	epheno object, as returned by ExpressionPhenoTest.
signatures	List with each element corresponding to a signature. The gene names in each signature must match those in epheno.
referenceSignature	If specified, the average fold change in each signature is compared to the average fold change in the signature referenceSignature.
testUpDown	If set to TRUE, bars corresponding to up and down-regulated genes are compared with those of referenceSignature separately. This argument is ignored if referenceSignature is not specified.
cex.text	Character expansion for the text indicating the P-values. Ignored if referenceSignature is missing.
alpha	Confidence levels for barplot error bars.
p.adjust.method	P-value adjustment method, passed on to p.adjust.
simulate.p.value	A logical indicating whether chi-square p-values should be computed by Monte Carlo simulation (passed on to chisq.test).
B	Integer specifying the number of replicates in the Monte Carlo simulation (passed on to chisq.test).
ylab	y-axis labels
ylim	y-axis limits
...	Other arguments to be passed on to boxplot.

Value

When a single signature is provided as input, a single plot assessing the association of that signature with all phenotype variables is created. If several signatures are provided, one separate plot is created for each phenotype variable.

Author(s)

Evarist Planet

Examples

```
#create epheno
data(epheno)

#construct two signatures
sign1 <- sample(featureNames(epheno))[1:20]
sign2 <- sample(featureNames(epheno))[1:15]
mySignature <- list(sign1,sign2)
names(mySignature) <- c('My first signature','My preferred signature')

#plot
barplotSignifSignatures(epheno[,'Relapse'],mySignature,alpha=0.05)
```

show-methods *Methods for Function show in Package 'methods'.*

Description

Methods for function `show` in Package 'methods'.

Methods

`signature(object = "AnnotatedDataFrame")` Will show an object of class `AnnotatedDataFrame`.

`signature(object = "ANY")` Will show an object of class `ANY`.

`signature(object = "classRepresentation")` Will show an object of class `classRepresentation`.

`signature(object = "container")` Will show an object of class `container`.

`signature(object = "epheno")` Will show an object of class `epheno`.

`signature(object = "eSet")` Will show an object of class `eSet`.

`signature(object = "genericFunction")` Will show an object of class `genericFunction`.

`signature(object = "gseaSignaturesSign")` Will show an object of class `gseaSignaturesSign`.

`signature(object = "gseaSignaturesVar")` Will show an object of class `gseaSignaturesVar`.

`signature(object = "gseaSignificanceSign")` Will show an object of class `gseaSignificanceSig`

`signature(object = "gseaSignificanceVar")` Will show an object of class `gseaSignificanceVar`.

`signature(object = "LargeDataObject")` Will show an object of class `LargeDataObject`.

`signature(object = "MethodDefinition")` Will show an object of class `MethodDefinition`.

`signature(object = "MethodSelectionReport")` Will show an object of class `MethodSelectionRep`

`signature(object = "MethodWithNext")` Will show an object of class `MethodWithNext`.

`signature(object = "MIAME")` Will show an object of class `MIAME`.

`signature(object = "namedList")` Will show an object of class `namedList`.

`signature(object = "ObjectsWithPackage")` Will show an object of class `ObjectsWithPackage`.

`signature(object = "oldClass")` Will show an object of class `oldClass`.

`signature(object = "ScalarCharacter")` Will show an object of class `ScalarCharacter`.

`signature(object = "ScalarObject")` Will show an object of class `ScalarObject`.

`signature(object = "signature")` Will show an object of class `signature`.

`signature(object = "TestResults")` Will show an object of class `TestResults`.

`signature(object = "traceable")` Will show an object of class `traceable`.

`signature(object = "Versioned")` Will show an object of class `Versioned`.

`signature(object = "Versions")` Will show an object of class `Versions`.

`signature(object = "VersionsNull")` Will show an object of class `VersionsNull`.

smoothCoxph *Plots the Cox proportional hazard smoothed by gene expression level.*

Description

Builds a plot showing how hazard behaves over different levels of expression of a given gene. Confidence intervals are also provided.

Usage

```
smoothCoxph(time, event, x, xlim, ylim, xlab, ylab, ...)
```

Arguments

time	variable where time to survival is stored.
event	variable where survival event is stored.
x	numeric containing the expression levels of a given gene.
xlim	xlim for the plot.
ylim	ylim for the plot.
xlab	xlab for the plot.
ylab	ylab for the plot.
...	other arguments that will be passed to plot.

Author(s)

David Rossell.

Examples

```
#load eset
data(eset)

#make plot
smoothCoxph(pData(eset)$Months2Relapse,pData(eset)$Relapse,exprs(eset)[25,])
```

summary.gseaSignificance *Obtain a data.frame with the pvalues and fdr for all signatures and variables of a gseaSignificanceSign or gseaSignificanceVar object.*

Description

Builds a data.frame object that can easily be written to a csv file containing the ES, NES, pval.ES, pval.NES and FDR.

Usage

```
summary.gseaSignificanceSign(object,...)
summary.gseaSignificanceVar(object,...)
```

Arguments

object	object of class gseaSignificanceSign or gseaSignificanceVar.
...	Arguments to be passed to summary.

Author(s)

Evarist Planet

References

Aravind Subramanian, (October 25, 2005) *Gene Set Enrichment Analysis*. www.pnas.org/cgi/doi/10.1073/pnas.0506580102

Examples

```
#for examples see the help file of gseaSigntaures: ?gseaSignatures
```

Index