Package 'MetaIntegrator'

September 29, 2016

Type Package

Title Meta-Analysis of Gene Expression Data

Version 1.0.3

Date 2016-09-27

Author Winston A. Haynes, Francesco Vallania, Aurelie Tomczak, Timothy Sweeney, Erika Bongen, Purvesh Khatri

Maintainer Winston A. Haynes <hayneswa@stanford.edu>

Description A pipeline for the meta-analysis of gene expression data. We have assembled several analysis and plot functions to perform integrated multi-cohort analysis of gene expression data (meta-analysis). Methodology described in: <http://biorxiv.org/content/early/2016/08/25/071514>.

License LGPL

Imports rmeta, multtest, ggplot2, parallel, Rmisc, reshape, gplots, Biobase, RMySQL, DBI, stringr, preprocessCore, GEOquery, GEOmetadb, RSQLite

Suggests BiocStyle, knitr, rmarkdown, RUnit, BiocGenerics

VignetteBuilder knitr

LazyData true

RoxygenNote 5.0.1

Depends R (>= 2.10)

URL http://biorxiv.org/content/early/2016/08/25/071514

NeedsCompilation no

Repository CRAN

Date/Publication 2016-09-29 08:02:32

R topics documented:

`backwardSearch` *Backward Search Function*

Description

Backward search is useful for reducing the size of the gene set in your filterObject. In general, backward search identifies a small set of genes with maximum ability to distinguish cases from controls.

backwardSearch is a method of optimizing a given set of significant genes to maximize discriminatory power, as measured by area under the ROC curve (AUC). The function works by taking a given set of genes (presumably a set that has been filtered for statistical significance), and iteratively removing one gene at a time, until the stopping threshold is reached. At each round, the gene whose removal contributes the greatest increase in weighted AUC is removed. Weight AUC is defined as the sum of the AUC of each dataset, times the number of samples in that dataset. The stopping threshold is in units of weighted AUC.

Usage

```
backwardSearch(metaObject, filterObject, backThresh = 0)
```

Arguments

`metaObject`	The metaObject from the main metaIntegrator function.
`filterObject`	An object matching the specifications for Filter
`backThresh`	Stopping threshold for the backward search. Default=0.

Details

The forwardSearch and backwardSearch functions are designed to assist in selection of gene sets optimized for discriminatory power. The selection of an optimized set is a non-convex problem, and hence both functions will yield gene sets that are only locally optimized (ie, they are not global optima). Both the forwardSearch and backwardSearch functions follow a greedy algorithm, either adding (or removing) genes that contribute the most (or the least) to the overall weighted AUC of the discovery datasets from the metaObject.

Both search functions allow a user to set a stopping threshold; the fundamental traseoff here will be sparsity of the returned gene set vs. overall discriminatory power. The default threshold is 0, such the functions will return the set of genes at which no gene could be added or removed for the forward or backward functions, respectively, and increase the weighted AUC.

Note that the weighted AUC returned during the function run is dependent on sample size; this was done (instead of a simple mean) so that the gene set discriminates the MOST SAMPLES, rather than being optimized for any particular dataset.

Value

A Filter object which has results from backward search

Author(s)

Timothy E. Sweeney

References

Sweeney et al., Science Translational Medicine, 2015

See Also

forwardSearch

Examples

```
#Run backward search to reduce the size of our filter results
backwardRes <- backwardSearch(tinyMetaObject, tinyMetaObject$filterResults[[1]], backThresh = -3)

#See the results
print(backwardRes$posGeneNames)
print(backwardRes$negGeneNames)
```

`calculateROC` *Calculate ROC Curve Statistics*

Description

Calculates receiver operating characteristic curve data, including AUC (using trapezoidal method). Takes only a vector of labels and a vector of predictions.

Usage

```
calculateROC(labels, predictions, AUConly = FALSE)
```

Arguments

`labels`	Vector of labels; must have exactly two unique values (ie, cases and controls).
`predictions`	Vector of predictions (for instance, test scores) to be evaluated for ability to separate the two classes. Must be exactly the same length as labels.
`AUConly`	Return all ROC values, or just the AUC.

Details

The code borrows its core ROC calculations from the ROCR package. AUC is calculated by the trapezoidal method. AUC standard errors are calculated according to Hanley's method.

Value

Assuming AUConly=F, returns a list of values:

`roc`	dataframe consisting of two columns, FPR and TPR, meant for plotting
`auc`	area under the curve
`auc.CI`	95% confidence interval for AUC

Author(s)

Timothy E. Sweeney

References

The code borrows its core ROC calculations from the ROCR package.

See Also

`calculateScore`, `rocPlot`

Examples

```
# expect an AUC near 0.5 with random test
labels <- c(rep(0, 500), rep(1, 500))
scores <- runif(1000)
calculateROC(labels, scores)

#With the real data, AUC should be around 0.85606
scoreResults <- calculateScore(tinyMetaObject$filterResults[[1]], tinyMetaObject$originalData[[1]])
rocRes <- calculateROC(predictions=scoreResults, labels=tinyMetaObject$originalData[[1]]$class)
print(rocRes$auc[[1]])
```

`calculateScore` *Calculate a signature Z-score for a set of genes in a single dataset*

Description

Given a gene set of interest, it is often desirable to summarize the expression of that gene set using a single integrated score. The `calculateScore` method calculates the geometric mean of the expression level of all positive genes, minus the geometric mean of the expression level of all negative genes. The resulting scores are then standardized within the given dataset, such that the output 'Z-score' has mean=0 and std. dev=1. Such a Z-score can then be used for classification, etc.

Usage

```
calculateScore(filterObject, datasetObject, suppressMessages=FALSE)
```

Arguments

`filterObject`	a MetaFilter object generated with `filterGenes()` containing the signature genes that will be used for Z-score calculation.
`datasetObject`	A Dataset object for which the signature score (Z-score) will be calculated. This vector would typically be added as `$score` column in `datasetObject$pheno`.
`suppressMessages`	Boolean value (TRUE/FALSE) about whether to display verbose output. Default: FALSE.

Details

The Z-score is based off of the geometric mean of expression. As such, negative expression values are not allowed. A dataset is thus always scaled by its minimum value + 1, such that the lowest value = 1. Any individual NANs or NAs are also set to 1. If a dataset does not have any information on a given gene, the entire gene is simply left out of the score. When run, the function will print to command line the number of genes used, and the number passed in.

Although mostly used internally, the function has been exported in case users want to compare multiple classes, etc., using the same Z-score as is used for producing two-class comparisons.

Value

A vector of Z-scores, of length `ncols(datasetObject$expr)` (and in the same order).

Author(s)

Timothy E. Sweeney, Winston A. Haynes

See Also

`filterGenes`

Examples

```
calculateScore(tinyMetaObject$filterResults[[1]], tinyMetaObject$originalData[[1]])
```

---

checkDataObject *Check for errors in objects used for analysis*

Description

Given an object to check, its objectType and the objectStage, the function `checkDataObject` looks for errors within Meta, Dataset, MetaAnalyis, or MetaFilter objects. It returns TRUE if the object passed error checking, FALSE otherwise, and it prints warning messages explaining failed checks.

Usage

```
checkDataObject(object, objectType, objectStage="")
```

Arguments

`object`	the object to be checked
`objectType`	one type of "Meta", "Dataset", "MetaAnalysis", "MetaFilter"
`objectStage`	if a `metaObject`, one of "Pre-Analysis", "Pre-Filter", or "Post-Filter". Otherwise: ""

Details

For `metaAnalysisObject` and `filterObject`, it makes sure that each entry within the object is 1) not NULL and 2) the correct type.

For `datasetObjects`, it makes sure that: 1) the entries are not null (except `$class`, which is permitted to be NULL) 2) the entries are the correct type and 3) the sample names (within `$pheno`, `$expr`, and `$class`) match 4) the probeIDs (within `$expr` and `$keys`) match.

For `metaObject`, it recursively checks the Dataset, MetaAnalysis, and MetaFilter objects contained within the `metaObject`.

The `objectStage` defines what entries a `metaObject` contains. Thus, "Pre-Analysis" `metaObjects` only contain `$originalData`. "Pre-Filter" `metaObjects` contain `$originalData`, `$metaAnalysis`, and `$leaveOneOutAnalysis`. "Post-Filter" `metaObjects` contain `$originalData`, `$metaAnalysis`, `$leaveOneOutAnalysis`, and `$filterResults`.

Value

checkPassed TRUE if passed error checking, FALSE otherwise

Prints warning messages explaining the portion of the error checking failed

Author(s)

Erika Bongen

Examples

```
# check a datasetObject
checkDataObject(tinyMetaObject$originalData$Whole.Blood.Study.1, "Dataset")

# check a metaObject before running the meta-analysis
checkDataObject(tinyMetaObject, "Meta", "Pre-Analysis")

# check a metaObject after running the meta-analysis with runMetaAnalysis()
checkDataObject(tinyMetaObject, "Meta", "Pre-Filter")

# check a metaObject after filtering the meta-analysis results with filterGenes()
checkDataObject(tinyMetaObject, "Meta", "Post-Filter")

# check a metaAnalysisObject
checkDataObject(tinyMetaObject$metaAnalysis, "MetaAnalysis")

# check a filterObject
checkDataObject(tinyMetaObject$filterResults[[1]], "MetaFilter")
```

---

ens_ensgID_table *ENSEMBL gene id table cache*

Description

Cached data to prevent cumbersome database connections.

---

ens_entrez_table *ENSEMBL entrez table cache*

Description

Cached data to prevent cumbersome database connections.

filterGenes *Filter out significant genes from meta-analysis results*

Description

After the Meta-Analysis results have been written to the `metaObject`, the results can be examined using different gene filtering criteria. This function will use the given filterParameter to select genes that fulfill the filter conditions. The function returns a modified version of the `metaObject` with results stored in `metaObject$filterResults`

Usage

```
filterGenes(metaObject, isLeaveOneOut=TRUE, effectSizeThresh=0, FDRThresh=0.05,
numberStudiesThresh=1, heterogeneityPvalThresh=0)
```

Arguments

`metaObject`	a Meta object which must have the `$originalData`, `$metaAnalysis` populated optional filterParameter:
`isLeaveOneOut`	Do leave-one-out analysis on discovery datasets (default: TRUE). Needs at least 2 datasets for discovery.
`FDRThresh`	FDR cutoff: a gene is selected, if it has a p-value less than or equal to the FDR cutoff (default: 0.05)
`effectSizeThresh`	a gene is selected, if the absolute value of its effect size is above this threashold (default: 0)
`numberStudiesThresh`	number of studies in which a selected gene has to be significantly up/down regulated (default: 1)
`heterogeneityPvalThresh`	heterogeneity p-value cutoff (filter is off by default: `heterogeneityPvalThresh = 0`). Genes with significant heterogeneity and, thus a significant (low) heterogeneity p-value, can be filtered out by using e.g.: `heterogeneityPvalThresh = 0.05` (removes all genes with heterogeneity p-value < 0.05)

Value

`metaObject`	A modified version of the input metaObject with an additional filterObject stored within `metaObject$filterResults`

Note

Use `checkDataObject(metaObject, "Meta", "Pre-Filter")` to make sure your metaObject has the right format for filtering after running the meta-analysis with `runMetaAnalysis()`.

Author(s)

Francesco Vallania

See Also

checkDataObject

Examples

```
# filter genes with default settings
#(false discovery rate cutoff of 5 percent and WITH leave-one-out analysis)
testMetaObject <- filterGenes(tinyMetaObject)
summarizeFilterResults(testMetaObject, getMostRecentFilter(testMetaObject))

# filter genes with false discovery rate of 1 percent and WITHOUT leave-one-out analysis
testMetaObject <- filterGenes(testMetaObject, FDRThresh = 0.01, isLeaveOneOut = FALSE)
summarizeFilterResults(testMetaObject, getMostRecentFilter(testMetaObject))
```

---

forestPlot *Compare effect sizes of a gene across all datasets in meta-analysis*

Description

A forest plot can be used to compare the expression values of a gene across different datasets. The size of the blue boxes is proportional to the number of samples in the study and light blue lines indicate the standard error of the effect sizes for each study (95% confidence interval). The summary effect size for all studies is indicated as yellow diamond below and the width of the diamond indicates the summary standard error.

Usage

```
forestPlot(metaObject, geneName)
```

Arguments

metaObject	a filtered metaObject, i.e. it needs to include a filterObject generated by the function filterGenes()
geneName	name of the gene for which the forest plot should be generated

Value

forest plot	Plot to compare effect sizes of a gene across datasets

Author(s)

Winston A. Haynes

See Also

filterGenes, runMetaAnalysis, violinPlot

Examples

```
# compare effect sizes of the Gene1 for all discovery datasets in tinyMetaObject
forestPlot(tinyMetaObject, geneName="Gene1")
```

`forwardSearch` *Forward Search Function*

Description

Forward search is useful for reducing the size of the gene set in your filterObject. In general, forward search identifies a small set of genes with maximum ability to distinguish cases from controls.

forwardSearch is a method of optimizing a given set of significant genes to maximize discriminatory power, as measured by area under the ROC curve (AUC). The function works by taking a given set of genes (presumably a set that has been filtered for statistical significance), and iteratively adding one gene at a time, until the stopping threshold is reached. At each round, the gene whose addition contributes the greatest increase in weighted AUC is added. Weight AUC is defined as the sum of the AUC of each dataset, times the number of samples in that dataset. The stopping threshold is in units of weighted AUC.

Usage

```
forwardSearch(metaObject, filterObject, yes.pos = NULL, yes.neg = NULL, forwardThresh = 0)
```

Arguments

`metaObject`	The metaObject from the main metaIntegrator function.
`filterObject`	An object matching the specifications for Filter
`yes.pos`	Optional- if passed, the forwardSearch will start with the genes in yes.pos and yes.neg (instead of starting from zero genes).
`yes.neg`	Optional- if passed, the forwardSearch will start with the genes in yes.pos and yes.neg (instead of starting from zero genes).
`forwardThresh`	Stopping threshold for the forward search. Default=0.

Details

The forwardSearch and backwardSearch functions are designed to assist in selection of gene sets optimized for discriminatory power. The selection of an optimized set is a non-convex problem, and hence both functions will yield gene sets that are only locally optimized (ie, they are not global optima). Both the forwardSearch and backwardSearch functions follow a greedy algorithm, either adding (or removing) genes that contribute the most (or the least) to the overall weighted AUC of the discovery datasets from the metaObject.

Both search functions allow a user to set a stopping threshold; the fundamental traseoff here will be sparsity of the returned gene set vs. overall discriminatory power. The default threshold is 0, such the functions will return the set of genes at which no gene could be added or removed for the forward or backward functions, respectively, and increase the weighted AUC.

Note that the weighted AUC returned during the function run is dependent on sample size; this was done (instead of a simple mean) so that the gene set discriminates the MOST SAMPLES, rather than being optimized for any particular dataset.

Value

A Filter object which has results from forward search

Author(s)

Timothy E. Sweeney

References

Sweeney et al., Science Translational Medicine, 2015

See Also

backwardSearch

Examples

```
#Run forward search to reduce the size of our filter results
forwardRes <- forwardSearch(tinyMetaObject, tinyMetaObject$filterResults[[1]], forwardThresh = 0)

#See the results
print(forwardRes$posGeneNames)
print(forwardRes$negGeneNames)
```

---

getGEOData *GEO download/processing through GEOquery*

Description

Creates MetaIntegrator formatted objects by downloading and formatting data from GEO.

Usage

```
getGEOData(gseVector, formattedNames = gseVector, ...)
```

Arguments

gseVector	a vector of GSE ids (each a string)
formattedNames	a vector of formatted names corresponding to the GSE ids. Default: gseVector
...	will pass additional parameters to getGEO, including destdir, which specifies download location

Value

a Pre-Analysis MetaObject containing the datasets loaded in $originalData

Author(s)

Francesco Vallania, Andrew Tam

---

getMostRecentFilter *Get name of most recent filter*

Description

Given a metaObject this function will look through $filterResults for the most recent filter used and return the filter name.

Usage

```
getMostRecentFilter(metaObject)
```

Arguments

metaObject	A meta object

Value

FilterLabel	Name of the most recent filter

Author(s)

Francesco Vallania

Examples

```
getMostRecentFilter(tinyMetaObject)
```

getSampleLevelGeneData *Extract gene-level data from a given data object*

Description

Given a datsetObject, and a set of target genes, this function will summarize probe-level data to gene-level data for the target genes. Returns a data frame with only the genes of interest, for each sample in the dataset.

Usage

```
getSampleLevelGeneData(datasetObject, geneNames)
```

Arguments

datasetObject	a Dataset object that is used to extract sample level data (At least, must have a $expr of probe-level data, and $keys of probe:gene mappings).
geneNames	A vector of geneNames

Details

Summarizes probe-level data to gene-level data, using the mean of the probes, according to the probe:gene mapping in the $keys item in the dataset object. This is done only for the genes in the filter object.

Value

Returns a data frame with expression levels of only the genes of interest, for each sample in the dataset.

Mostly used internally, but has been exposed to the user to allow advanced functionality on external datasets if desired.

Author(s)

Timothy E. Sweeney

Examples

```
sampleResults <- getSampleLevelGeneData(datasetObject=tinyMetaObject$originalData[[1]],
geneNames=c(tinyMetaObject$filterResults[[1]]$posGeneNames,
tinyMetaObject$filterResults[[1]]$negGeneNames))
```

`heatmapPlot` *Generates a heatmap with effect sizes for all genes which pass a filter in all measured diseases*

Description

Generates a heatmap with effect sizes for all genes which pass a filter in all measured diseases

Usage

```
heatmapPlot(metaObject, filterObject, colorRange = c(-1, 1))
```

Arguments

`metaObject`	a Meta object which must have the $originalData, $metaAnalysis populated
`filterObject`	a MetaFilter object containing the signature genes that will be used for the heatmap
`colorRange`	a vector of length two with the minimum and maximum values for the heatmap colors. (default: c(-1,1))

Value

Generates a heatmap with effect sizes for all genes which pass a filter

Author(s)

Winston A. Haynes

Examples

```
heatmapPlot(tinyMetaObject, tinyMetaObject$filterResults[[1]])
```

`MetaIntegrator` *MetaIntegrator package for meta-analysis of gene expression data*

Description

The package comprises several analysis and plot functions to perform integrated multi-cohort analysis of gene expression data (meta-analysis).

Package:	metaIntegrator_public
Type:	Package
Version:	1.0
Date:	2015-02-25
License:	LGPL

For detailed documentation of functions and use cases read: `vignette(MetaIntegrator)`.

Details

The advent of the gene expression microarray has allowed for a rapid increase in gene expression studies. There is now a wealth of publically available gene expression data available for re-analysis. An obvious next step to increase statistical power in detecting changes in gene expression associated with some condition is to aggregate data from multiple studies.

The MetaIntegrator package will perform a DerSimonian & Laird random-effects meta-analysis for each gene (not probeset) between all target studies between cases and controls; it also performs a Fischer's sum-of-logs method on the same data, and requires that a gene is significant by both methods. The resulting p-values are False discovery rate (FDR) corrected to q-values, and will evaluate the hypothesis of whether each gene is differentially expressed between cases and controls across all studies included in the analysis.

The resulting list of genes with significantly different expression between cases and controls can be used for multiple purposes, such as (1) a new diagnostic or prognostic test for the disease of interest, (2) a better understanding of the underlying biology, (3) identification of therapeutic targets, and multiple other applications.

Our lab has already used these methods in a wide variety of diseases, including organ transplant reject, lung cancer, neurodegenerative disease, and sepsis (Khatri et al., J Exp Med 2013; Chen et al, Cancer Res 2014; Li et al., Acta Neur Comm 2014; Sweeney et al, Sci Trans Med 2015).

Author(s)

Winston A. Haynes, Francesco Vallania, Aurelie Tomczak, Timothy E. Sweeney, Erika Bongen, Purvesh Khatri

Maintainer: Winston A. Haynes <hayneswa@stanford.edu>

References

Sweeney et al., Science Translational Medicine, 2015

Khatri P et al. J Exp. Med. 2013

See Also

vignette(MetaIntegrator)

Examples

```
#Run a meta analysis.
#   maxCores is set to 1 for package guideline compliance.
# For personal purposes, leave parameter un-set.
runMetaAnalysis(tinyMetaObject, maxCores=1)

#### a standard meta-analysis would follow this work flow: ####

# make input metaObjects from individual GEO datasetObjects
metaObject = list()
```

```
metaObject$originalData <- tinyMetaObject$originalData
# make test datasetObject
datasetObject1 <- tinyMetaObject$originalData$Whole.Blood.Study.1

# run the meta-analysis
metaObject <- runMetaAnalysis(metaObject, maxCores=1)

# select significant genes (default parameter)
metaObject <- filterGenes(metaObject)

# print a meta-analysis result summary for selected genes
summarizeFilterResults(metaObject, getMostRecentFilter(metaObject))

# use selected genes to generate a violin plot
violinPlot(metaObject$filterResults$FDR0.05_es0_nStudies1_looaTRUE_hetero0, datasetObject1,
labelColumn = 'group')

# use selected genes to generate a ROC plot
rocPlot(metaObject$filterResults$FDR0.05_es0_nStudies1_looaTRUE_hetero0, datasetObject1)

# generate a forest plot for a gene of interest with forestPlot(metaObject, geneName)
forestPlot(metaObject, "Gene27")
```

`regressionPlot` *Generate a plot which draws a regrssion line between the Meta Score and a continuous variable phenotype.*

Description

Generate a plot which draws a regrssion line between the Meta Score and a continuous variable phenotype.

Usage

```
regressionPlot(filterObject, datasetObject,
  continuousVariableColumn = "continuous",
  formattedVariableName = "Continuous Variable")
```

Arguments

`filterObject`	a MetaFilter object containing the signature genes that will be used for the z-score calculation
`datasetObject`	a Dataset object (typically independent validation dataset) for comparison in a regression plot
`continuousVariableColumn`	the label of the column in $pheno that sepecifies the continuous variable to compare (default: 'continuousVariableColumn')
`formattedVariableName`	label which will be used on the x-axis on the plot

Value

Returns a regression plot as ggplot2 plot object

Author(s)

Winston A. Haynes

`rocPlot` *Plot ROC Curve for a Dataset*

Description

rocPlot will plot an ROC curve (and return the AUC) that describes how well a gene signature (as defined in a `filterObject`) classifies groups in a dataset (in the form of a `datasetObject`).

Usage

```
rocPlot(filterObject, datasetObject, title = datasetObject$formattedName)
```

Arguments

`filterObject`	a MetaFilter object containing the signature genes that will be used for calculation of the ROC plot.
`datasetObject`	a Dataset object for group comparison in the ROC plot. (At least, must have a `$expr` of probe-level data, `$keys` of probe:gene mappings, and `$class` of two-class labels.)
`title`	Title for the ROC plot.

Details

Evaluates the ability of a given gene set to separate two classes. The gene set is evaluated as a Z-score of the difference in means between the positive genes and the negative genes (see calculateScore). Returns a standard ROC plot, plus AUC with 95% CI (calculated according to Hanley method).

Value

Returns a ggplot2 plot object

Author(s)

Timothy E. Sweeney

See Also

`calculateScore`, `calculateROC`

Examples

```
rocPlot(tinyMetaObject$filterResults[[1]], tinyMetaObject$originalData[[1]])
```

`runMetaAnalysis` *Run the meta-analysis algorithm*

Description

Given a `metaObject` with `$originalData` populated this function will run the meta-analysis algorithm.

It returns a modified version of the `metaObject` with the meta-analysis results written into `metaObject$metaAnalysis` and the results of the leave-one-out analysis into `metaObject$leaveOneOutAnalysis`

Usage

```
runMetaAnalysis(metaObject, runLeaveOneOutAnalysis= TRUE, maxCores=Inf)
```

Arguments

`metaObject`	a metaObject which must have `metaObject$originalData` populated with a list of `datasetObjects` that will be used for discovery
`runLeaveOneOutAnalysis`	TRUE to run leave one out analysis, FALSE otherwise (default: TRUE)
`maxCores`	maximum number of cores to use during analysis (default: Inf)

Value

`metaObject`	modified version of the `metaObject` with `$metaAnalysis` and `$leaveOneOutAnalysis` populated

Warning

To make sure the input is correctly formatted, the input `metaObject` should be checked with `checkDataObject(metaObject, "Meta", "Pre-Analysis")` before starting the meta-analysis.

Author(s)

Francesco Vallania

See Also

`checkDataObject`

Examples

```
#Run a meta analysis.
# maxCores is set to 1 for package guideline compliance.
# For personal purposes, leave parameter un-set.
runMetaAnalysis(tinyMetaObject, maxCores=1)
```

`summarizeFilterResults`

Summarize the filtered analysis results

Description

Given a `metaObject` and the name of the `filterObject` of interest, this function will print a summary style message about genes that passed the filtering step using the function `filterGenes()` and return a `dataFrame` that contains the `$pooledResults` information for each gene which passed the filter.

Usage

```
summarizeFilterResults(metaObject, metaFilterLabel)
```

Arguments

`metaObject`	the metaObject that contains the `filterObject` of interest
`metaFilterLabel`	the name of a `filterObject` generated with the function `filterGenes()`

Value

`dataFrame`	Data frame, which contains `$pooledResults` information for each gene which passed the filter

Author(s)

Francesco Vallania

See Also

`filterGenes`

Examples

```
# filter genes with default settings
# false discovery rate cutoff of 5 percent and WITH leave-one-out analysis
testMetaObject <- filterGenes(tinyMetaObject)
summarizeFilterResults(testMetaObject, getMostRecentFilter(testMetaObject))
```

tinyMetaObject *A Tiny MetaObject*

Description

This is a minimal working example of a MetaObject. This object is primarily used for example function calls and visualizations

Author(s)

Winston A. Haynes

ucsc_genbank_table *UCSC genbank table cache*

Description

Cached data to prevent cumbersome database connections.

ucsc_refseq_table *UCSC refseq table cache*

Description

Cached data to prevent cumbersome database connections.

violinPlot *Compare groups within a single dataset in a violin plot*

Description

Given a `filterObject` and a `datasetObject` this function will use the selected genes of the `filterObject` to calculate and compare the z-scores of the groups (e.g. cases vs. controls) from the `datasetObject` by generating a violin plot. A violin plot is similar to a box plot, except the width of each violin is proportional to the density of points. `violinPlot()` is commonly used to validate a gene signature in an independent dataset.

Usage

```
violinPlot(filterObject, datasetObject, labelColumn="label")
```

Arguments

filterObject	a MetaFilter object containing the signature genes that will be used for the z-score calculation
datasetObject	a Dataset object (typically independent validation dataset) for group comparison in a violin plot
labelColumn	the label of the column in $pheno that sepecifies the groups to compare, typically case or control (default: 'label')

Details

The z-score is based off of the geometric mean of expression. As such, negative expression values are not allowed. A dataset is thus always scaled by its minimum value + 1, such that the lowest value = 1. Any individual NANs or NAs are also set to 1. If a dataset does not have any information on a given gene, the entire gene is simply left out of the score.

Value

Returns a violin plot as ggplot2 plot object

Author(s)

Winston A. Haynes

See Also

filterGenes, runMetaAnalysis

Examples

```
violinPlot(tinyMetaObject$filterResults$pValueFDR0.05_es0_nStudies1_looaTRUE_hetero0,
tinyMetaObject$originalData$Whole.Blood.Study.1, labelColumn="group")
```

Index