# Package 'MetaIntegrator'

July 23, 2019

**Type** Package

**Title** Meta-Analysis of Gene Expression Data

**Version** 2.1.1

**Date** 2019-07-23

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

**Maintainer** Aditya M. Rao <adityamr@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

**biocViews**

**Imports** BiocManager, rmeta, multtest, ggplot2, parallel, Rmisc, gplots, Biobase, RMySQL, DBI, stringr, preprocessCore, GEOquery, GEOmetadb, RSQLite, data.table, ggpubr, ROCR, zoo, pracma, COCONUT, Metrics, manhattanly, DT, pheatmap, plyr, boot, dplyr, reshape2, rmarkdown, AnnotationDbi, HGNChelper, magrittr, readr, plotly, httpuv

**Suggests** BiocStyle, knitr, RUnit, BiocGenerics, snplist

**VignetteBuilder** knitr

**LazyData** true

**RoxygenNote** 6.1.1

**Depends** R (>= 3.4)

**Encoding** UTF-8

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

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2019-07-23 14:52:07 UTC

# 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 tradeoff 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$filterResults[[1]], backThresh
#See the results
print(backwardRes$posGeneNames)
 print(backwardRes$negGeneNames)
```

---

## calcMetaPower *Calculates the statistical power of a random effects meta-analysis*

---

### Description

Calculates the statistical power of a random effects meta-analysis based on the methods described by Valentine et al. 2010, J of Educational and Behavioral Studies.

### Usage

```
calcMetaPower(es, avg_n, nStudies, hg, tail=2)
```

### Arguments

| | |
|---|---|
| `es` | effect size you're trying to detect (e.g. 0.6) |
| `avg_n` | the average sample size of each GROUP in each STUDY (e.g. 10) |
| `nStudies` | the number of studies you put in the meta-analysis (aka Discovery cohort) (e.g. 5) |
| `hg` | heterogeneity, (".33" for small, "1" for moderate, & "3" for large) (e.g. 0.33) |
| `tail` | whether you have a one tail or two tail p-value |

### Details

Based on the paper by Valentine et al.: JC Valentine, TD Pigott, and HR Rothstein. How Many Studies Do You Need? A Primer on Statistical Power for Meta-Analysis J of Educational and Behavioral Statistics April 2010 Vol 35, No 2, pp 215-247

The code itself is adapted from a blog post by Dan Quintana, Researcher at Oslo University in Biological Psychiatry On the website Towards Data Science, July 2017

https://towardsdatascience.com/how-to-calculate-statistical-power-for-your-meta-analysis-e108ee586ae8

`avg_n` is the average number people in each group in each study, so if you have 4 studies, and each study compared 10 cases and 10 controls, then `avg_n` = 10.

NOTE: THIS CODE DOES NOT TAKE MULTIPLE HYPOTHESIS TESTING INTO ACCOUNT IT ASSUMES P< 0.05

For clarity, avg_n is the average number people in each group in each study, so if you have 4 studies, and each study compared 10 cases and 10 controls, then avg_n = 10.

### Value

Statistic `Power` of the random effects meta-analysis described. Most statisticians want a statistical power of at least 0.8, which means that there is an 80 that if there is a true effect, you will detect it.

### Examples

```
# effect size =0.7
# 10 samples on average in each group in each study
# 5 studies included in meta-analysis
# low heterogeneity (0.33)
calcMetaPower(es=0.7, avg_n=10, nStudies=5, hg=0.33)
```

---

## `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$filterResults[[1]], tinyMetaObject$originalDat
rocRes <- calculateROC(predictions=scoreResults, labels=tinyMetaObject$originalData[[1]]$cla
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

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")
```

---

`classFunction` *Helper function to build the class vector*

---

## Description

Helper function to build the class vector

## Usage

```
classFunction(datasetObject, column, diseaseTerms)
```

## Arguments

| | |
|---|---|
| `datasetObject` | the Dataset object to build a class vector for |
| `column` | column from the $pheno slot to look for the disease terms |
| `diseaseTerms` | a list of terms identifying the disease samples |

## Details

Based on a defined set of disease terms, builds a class vector.

## Value

returns a Dataset object that has a class vector inserted

## Author(s)

Winston A. Haynes

## Examples

```
classObj <- classFunction(tinyMetaObject$originalData$Whole.Blood.Study.1,
   column="group", diseaseTerms=c("Disease"))
```

`cleanUpPheno` *Automatic preprocessing of $pheno dataframe*

## Description

Takes a Dataset object and:

## Usage

```
cleanUpPheno(myDataset)
```

## Arguments

| | |
|---|---|
| `myDataset` | a `datasetObject` that contains unprocessed `$pheno` |

## Value

myDataset a `datasetObject` that contains processed `$pheno` and original unprocessed `$rawPheno`

## Author(s)

Erika Bongen

## Examples

```
## Not run:
# Download and automatically preprocess pheno
gse53195 = getGEOData("GSE53195")
gse53195 = gse53195$originalData$GSE53195
View(gse53195$pheno) # Original $pheno
gse53195 = cleanUpPheno(gse53195)
View(gse53195$rawPheno) # Original $pheno
View(gse53195$pheno) # Preprocessed $Pheno

## End(Not run)
```

`coconutMetaIntegrator` *A wrapper function to run COCONUT on the MetaIntegrator objects.*

## Description

A wrapper function to run COCONUT on the MetaIntegrator objects.

### Usage

```
coconutMetaIntegrator(metaObject, ...)
```

### Arguments

| | |
|---|---|
| metaObject | a MetaIntegrator formatted Meta object. |
| ... | pass along arguments to COCONUT |

### Value

Results from COCONUT analysis on the MetaIntegrator object

### Author(s)

Winston A. Haynes

---

## 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 |
| `isLeaveOneOut` | Do leave-one-out analysis on discovery datasets (default: TRUE). Needs at least 2 datasets for discovery. |
| `effectSizeThresh` | a gene is selected, if the absolute value of its effect size is above this threshold (default: 0) |
| `FDRThresh` | FDR cutoff: a gene is selected, if it has a p-value less than or equal to the FDR cutoff (default: 0.05) |
| `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

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, boxColor = "blue", whiskerColor = "lightblue",
zeroLineColor = "black", summaryColor = "orange", textColor = "red")
```

### 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 |
| `boxColor` | desired color for the box (default: "blue") |
| `whiskerColor` | desired color for the whiskers (default: "lightblue") |
| `zeroLineColor` | desired color for the line indicating 0 (default: "black") |
| `summaryColor` | desired color for the diamond representing the summary effect size (default: "orange") |
| `textColor` | desired color for the text of the dataset names (default: "red") |

### Value

Plot to compare effect sizes of a gene across datasets

### Author(s)

Winston A. Haynes, Jiaying Toh

### 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 tradeoff 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)
```

## geneSymbolCorrection *Correct/update gene symbols in a metaObject*

### Description

The gene symbols in gene expression data are sometimes outdated or incorrect, so this function goes through your metaObject and updates the symbols based on the HGNChelper package, as well as correcting some other known issues.

### Usage

```
geneSymbolCorrection(metaObject)
```

### Arguments

| | |
|---|---|
| metaObject | your metaObject |

### Value

A modified version of the input metaObject with updated gene symbols for each dataset in `metaObject$originalData`

### Author(s)

Aditya M. Rao

### Examples

```
tinyMetaObject = geneSymbolCorrection(tinyMetaObject)
```

## getGEOData *GEO download/processing through GEOquery*

### Description

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

### Usage

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

### Arguments

| | |
|---|---|
| gseVector | a vector of GSE ids (each a string) |
| formattedNames | a vector of formatted names corresponding to the GSE ids. Default: gseVector |
| qNorm | perform quantile normalization of expression data within a dataset or not. Default: FALSE |
| ... | will pass additional parameters to getGEO, including destdir, which specifies download location |

### Details

Note: if you get the error "Error: Couldn't find driver MySQL" then just library(RMySQL) and then re-run getGEOData

### Value

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

### Author(s)

Francesco Vallania, Andrew Tam, Ravi Shankar, Aditya M. Rao

---

## 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

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, Winston A. Haynes

### Examples

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

`ggForestPlot` *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 area of the blue boxes is proportional to the number of samples in the study and black lines indicate the standard error of the effect sizes for each study (by default the 95% confidence interval). The summary effect size for all studies is indicated as an orange diamond below and the width of the diamond indicates the summary standard error.

## Usage

```
ggForestPlot(metaObject, genes, confLevel = 0.95, facetCols = NULL,
  facetScales = "free_x", boxScales = c(6, 16))
```

## Arguments

| | |
|---|---|
| `metaObject` | a filtered metaObject, i.e. it needs to include a `filterObject` generated by the function `filterGenes()` |
| `genes` | character vector containing the genes for which the forest plot should be generated |
| `confLevel` | confidence level |
| `facetCols` | integer that specifies how many columns are going to be used for the plot |
| `facetScales` | same as ggplot's facet_wrap: should Scales be fixed ("fixed", the default), free ("free"), or free in one dimension ("free_x", "free_y") |
| `boxScales` | a numeric vector of length 2 providing scaling factors for the plot. Specifies minimum and maximum size. |

## Value

ggplot2 Plot comparing effect sizes of a gene across datasets

## Author(s)

Winston A. Haynes, Jiaying Toh, Michele Donato

## See Also

`filterGenes`, `runMetaAnalysis`, `violinPlot`

## Examples

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

`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),
  geneOrder = FALSE, datasetOrder = FALSE, displayPooled = TRUE,
  useFormattedNames = TRUE)
```

## 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)) |
| `geneOrder` | FALSE if the genes should be ordered by pooled effect size in this datasets. Otherwise, the ordered names of the genes. (default: FALSE) |
| `datasetOrder` | FALSE if the datasets should be ordered alphabetically. Otherwise, the ordered names of the datasets (default: FALSE) |
| `displayPooled` | TRUE if the pooled effect sizes should be displayed. (default: TRUE) |
| `useFormattedNames` | TRUE if the formatted datasetNames should be displayed. (default: TRUE) |

## 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]])
```

## immunoStatesDecov *immunoStates deconvolution analysis on MetaIntegrator object(s)*

### Description

immunoStates deconvolution analysis on MetaIntegrator object(s)

### Usage

```
immunoStatesDecov(metaObject)
```

### Arguments

| | |
|---|---|
| metaObject | a MetaIntegrator formatted Meta object. |

### Value

Results from immunoStates on the MetaIntegrator object are stored in $immunoStates of the original Meta object

### Author(s)

Francesco Vallania

### Examples

```
## Not run:
# Example won't work on tinyMetaObject because it requires real gene names
# Download the needed datasets for processing.
sleData <- getGEOData(c("GSE11909","GSE50635", "GSE39088"))

# Run immunoStates
immunoStatesEstimates <- immunoStateDecov(sleData)

## End(Not run)
```

## immunoStatesGenePropCorr *Correct gene expression using cell proportions from immunoStates*

### Description

Correct gene expression using cell proportions from immunoStates

### Usage

```
immunoStatesGenePropCorr(metaObject)
```

### Arguments

`metaObject` a MetaIntegrator formatted Meta object.

### Value

Results from immunoStates gene proportion correction on the MetaIntegrator object are stored in $iScorrExp of the original Meta object

### Author(s)

Francesco Vallania copyright by Francesco Vallania

### Examples

```
## Not run:
# Example won't work on tinyMetaObject because it requires real gene names
# Download the needed datasets for processing.
sleData <- getGEOData(c("GSE11909","GSE50635", "GSE39088"))

# Run immunoStates
immunoStatesCorrected <- immunoStateGenePropCorr(sleData)

## End(Not run)
```

---

## `immunoStatesMatrix` *immunoStates basis matrix*

---

### Description

immunoStates basis matrix

### Usage

```
data("immunoStatesMatrix")
```

---

## `immunoStatesMeta` *immunoStates deconvolution analysis on MetaIntegrator object(s)*

---

### Description

Run immunoStates and load the results into $originalData for running meta-analysis on the cell proportion estimates.

### Usage

```
immunoStatesMeta(metaObject)
```

### Arguments

| | |
|---|---|
| metaObject | a MetaIntegrator formatted Meta object. |

### Value

Results from immunoStates stored in $originalData

### Author(s)

Francesco Vallania

### Examples

```
## Not run:
# Example won't work on tinyMetaObject because it requires real gene names
# Download the needed datasets for processing.
sleData <- getGEOData(c("GSE11909","GSE50635", "GSE39088"))

# Run immunoStates
immunoStatesEstimates <- immunoStateMeta(sleData)

## End(Not run)
```

---

## imputeSex *Imputes biological sex of each sample in a Dataset object*

---

### Description

Imputes biological sex of each sample in a Dataset object

### Usage

```
imputeSex(myDataset, femGenes = NULL, malGenes = NULL)
```

### Arguments

| | |
|---|---|
| myDataset | datasetObject |
| femGenes | vector of gene symbols of genes higher expressed in females. Defaults to NULL |
| malGenes | vector of gene symbols of genes higher expressed in males. Defaults to NULL |

### Details

Imputes the sex of each sample in a Dataset object by performing K means clustering. If genes higher expressed in females (femGenes) and genes higher expressed in males (malGenes) are not supplied, then clustering is performed on a default set of known X-escape genes (Tukiainen et al. 2017 Nature) and Y-chromosome genes. Genes were chosen as a subset of the immune Sex Expression Signature (iSEXS) (Bongen et al. In Prep.)

Known X-Escape genes: "XIST","RPS4X","CD40LG","ZRSR2","EFHC2","CA5B","ZFX","EIF1AX","CA5BP1","UBA1"

Y-Chromosome genes: "KDM5D","RPS4Y1","EIF1AY","USP9Y","DDX3Y","UTY","PRKY","ZFY","TMSB4Y"

## Value

a vector indicating whether each sample is classified as "male" or "female"

## Author(s)

Erika Bongen

## Examples

```
# Add sex labels to your dataset of choice
## Not run:
myDatasets = getGEOData(c("GSE13485","GSE17156","GSE19442"))
myDatasets$originalData$GSE13485$pheno$sex = imputeSex(myDatasets$originalData$GSE13485)
myDatasets$originalData$GSE13485$pheno$sex

## End(Not run)
```

---

`lincsBaitCorr` *Run Shane's LINCS bait-based correlation on MetaIntegrator*

---

## Description

LINCS Bait Corr finds perturbagens similar to a set of interest, called baits. It searches within a defined sub space of relevant genes, usually a disease signature See below for an example that recreates the work we did to find the antiviral drugs

## Usage

```
lincsBaitCorr(metaObject, filterObject, dataset = "CP", baits,
  just_clin = F, hit.number.hm = 20, hm_baits = T,
  direction = "aggravate", bait_type = NULL)
```

## Arguments

| | |
|---|---|
| `metaObject` | a Meta object which must have the $originalData populated |
| `filterObject` | a MetaFilter object containing the signature genes that will be used for calculating the score |
| `dataset` | The LINCS dataset to use. One of "CP" (drugs),"SH" (shRNA),"OE" (overexpression), "LIG" (ligands),"MUT" (mutants) (default: CP) |
| `baits` | vector containing names of the baits being used (relevant drugs, shRNAs, etc.). See example. |
| `just_clin` | only consider clinically relevant results (default: FALSE) |
| `hit.number.hm` | How many hits to show in a heatmap (default: 20) |
| `hm_baits` | whether or not to include the baits in the heatmap (default: FALSE) |

direction — one of "reverse", "aggravate", or "absolute" (default: "reverse") for whether you want to reverse the signature, aggravate it, or just want the top absolute hits.

bait_type — The LINCS dataset where the baits come from. One of "CP" (drugs),"SH" (shRNA),"OE" (over-expression), "LIG" (ligands),"MUT" (mutants), or NULL (don't specify) (default:NULL)

## Value

The full list of correlations as well as the dataframe with the expression of the top hits. Also generates the heatmap of the top hits.

## Examples

```
## Not run:
####### DATA SETUP ##########
# Example won't work on tinyMetaObject because it requires real gene names
# Download the needed datasets for processing.
sleData <- getGEOData(c("GSE11909","GSE50635", "GSE39088"))

#Label classes in the datasets
sleData$originalData$GSE50635 <- classFunction(sleData$originalData$GSE50635,
  column = "subject type:ch1", diseaseTerms = c("Subject RBP +", "Subject RBP -"))
sleData$originalData$GSE11909_GPL96 <- classFunction(sleData$originalData$GSE11909_GPL96,
   column = "Illness:ch1", diseaseTerms = c("SLE"))
sleData$originalData$GSE39088 <- classFunction(sleData$originalData$GSE39088,
   column= "disease state:ch1", diseaseTerms=c("SLE"))
 #Remove the GPL97 platform that was downloaded
sleData$originalData$GSE11909_GPL97 <- NULL

#Run Meta-Analysis
sleMetaAnalysis <- runMetaAnalysis(sleData, runLeaveOneOutAnalysis = F, maxCores = 1)

#Filter genes
sleMetaAnalysis <- filterGenes(sleMetaAnalysis, isLeaveOneOut = F,
   effectSizeThresh = 1, FDRThresh = 0.05)
####### END DATA SETUP ##########

#Note: these are note relevant baits for SLE, just examples
 lincsBaitCorr(metaObject = sleMetaAnalysis, filterObject = sleMetaAnalysis$filterResults[[1
   dataset = "CP", baits = c("NICLOSAMIDE","TYRPHOSTINA9","DISULFIRAM","SU4312","RESERPINE")

## End(Not run)
```

---

# lincsCorrelate *Run Shane's LINCS Correlate on MetaIntegrator*

---

## Description

Run Shane's LINCS Correlate on MetaIntegrator

### Usage

```
lincsCorrelate(metaObject, filterObject, dataset = "CP",
  hit.number.hm = 20, direction = "reverse", cor.method = "pearson",
  drop.string = NULL, just_clin = F, show_clin = F, gene_ann = F)
```

### Arguments

| | |
|---|---|
| metaObject | a Meta object which must have the $originalData populated |
| filterObject | a MetaFilter object containing the signature genes that will be used for calculating the score |
| dataset | The LINCS dataset to use. One of "CP" (drugs),"SH" (shRNA),"OE" (overexpression), "LIG" (ligands),"MUT" (mutants) (default: CP) |
| hit.number.hm | How many hits to show in a heatmap (default: 20) |
| direction | one of "reverse", "aggravate", or "absolute" (default: "reverse") for whether you want to reverse the signature, aggravate it, or just want the top absolute hits. |
| cor.method | method to use for correlation (pearson or spearman) (default: "pearson") |
| drop.string | lets you include a string to drop drugs that contain a regular expression. Useful for getting rid of screening hits. One useful option is "^BRD", which gets rid of all of the Broad screening hits that aren't characterized. (default: NULL) |
| just_clin | only consider clinically relevant results (default: FALSE) |
| show_clin | Generate a list of clinically relevant results (default: FALSE) |
| gene_ann | whether to annotate genes (default: FALSE) |

### Value

The full list of correlations as well as the dataframe with the expression of the top hits. Also generates the heatmap of the top hits.

### Examples

```
## Not run:
####### DATA SETUP ##########
# Example won't work on tinyMetaObject because it requires real gene names
# Download the needed datasets for processing.
sleData <- getGEOData(c("GSE11909","GSE50635", "GSE39088"))

#Label classes in the datasets
sleData$originalData$GSE50635 <- classFunction(sleData$originalData$GSE50635,
  column = "subject type:ch1", diseaseTerms = c("Subject RBP +", "Subject RBP -"))
sleData$originalData$GSE11909_GPL96 <- classFunction(sleData$originalData$GSE11909_GPL96,
   column = "Illness:ch1", diseaseTerms = c("SLE"))
sleData$originalData$GSE39088 <- classFunction(sleData$originalData$GSE39088,
   column= "disease state:ch1", diseaseTerms=c("SLE"))
 #Remove the GPL97 platform that was downloaded
sleData$originalData$GSE11909_GPL97 <- NULL
```

```
#Run Meta-Analysis
sleMetaAnalysis <- runMetaAnalysis(sleData, runLeaveOneOutAnalysis = F, maxCores = 1)

#Filter genes
sleMetaAnalysis <- filterGenes(sleMetaAnalysis, isLeaveOneOut = F,
   effectSizeThresh = 1, FDRThresh = 0.05)
####### END DATA SETUP ##########

 lincsCorrelate( metaObject = sleMetaAnalysis, filterObject = sleMetaAnalysis$filterResults[
    dataset = "CP", direction = "reverse")

## End(Not run)
```

---

`lincsTools` *Run Shane's LINCS Tools on MetaIntegrator*

---

## Description

Run Shane's LINCS Tools on MetaIntegrator

## Usage

```
lincsTools(metaObject, filterObject, report.out.folder,
  hit.number.hm = 10, hit.number.tbl = 10, resize = F,
  reportTitle = "lincsReport")
```

## Arguments

| | |
|---|---|
| `metaObject` | a Meta object which must have the $originalData populated |
| `filterObject` | a MetaFilter object containing the signature genes that will be used for calculating the score |
| `report.out.folder` | Directory where a report with all figures and tables will be generated. |
| `hit.number.hm` | How many hits to show in a heatmap (default:10) |
| `hit.number.tbl` | How many hits to show in a displayed table (default:10) |
| `resize` | Whether to resize tables in the way Purvesh prefers for figures (default: FALSE) |
| `reportTitle` | file prefix for report outputs (default: "lincsReport") |

## Value

LINCS report for the data

## Examples

```
## Not run:
####### DATA SETUP ##########
# Example won't work on tinyMetaObject because it requires real gene names
# Download the needed datasets for processing.
sleData <- getGEOData(c("GSE11909","GSE50635", "GSE39088"))

#Label classes in the datasets
sleData$originalData$GSE50635 <- classFunction(sleData$originalData$GSE50635,
  column = "subject type:ch1", diseaseTerms = c("Subject RBP +", "Subject RBP -"))
sleData$originalData$GSE11909_GPL96 <- classFunction(sleData$originalData$GSE11909_GPL96,
   column = "Illness:ch1", diseaseTerms = c("SLE"))
sleData$originalData$GSE39088 <- classFunction(sleData$originalData$GSE39088,
   column= "disease state:ch1", diseaseTerms=c("SLE"))
 #Remove the GPL97 platform that was downloaded
sleData$originalData$GSE11909_GPL97 <- NULL

#Run Meta-Analysis
sleMetaAnalysis <- runMetaAnalysis(sleData, runLeaveOneOutAnalysis = F, maxCores = 1)

#Filter genes
sleMetaAnalysis <- filterGenes(sleMetaAnalysis, isLeaveOneOut = F,
   effectSizeThresh = 1, FDRThresh = 0.05)
####### END DATA SETUP ##########

# Run immunoStates
 lincsTools(influenzaMeta, influenzaMeta$filterResults$FDR0.05_es0_nStudies4_looaTRUE_heterc

## End(Not run)
```

---

`manhattanPlot` *Generates a Manhattan plot with effect size FDR as y-axis*

---

## Description

Generates a Manhattan plot with effect size FDR as y-axis

## Usage

```
manhattanPlot(metaObject)
```

## Arguments

| | |
|---|---|
| `metaObject` | a Meta object which must have meta-analysis run |

## Value

Generates a Manhattan plot with effect size FDR as y-axis

### Author(s)

Winston A. Haynes

---

`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 publicly 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

```
## Not run:
#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")

## End(Not run)
```

---

`multiplePRCPlot` *Generate a plot with multiple PRC curves*

---

## Description

for each dataset in the metaObject, prcPlot will return a ggplot of a Precision-Recall curve (and return the AUPRC) that describes how well a gene signature (as defined in a `filterObject`) classifies groups in a dataset (in the form of a `datasetObject`).

## Usage

```
multiplePRCPlot(metaObject, filterObject, title = NULL,
  legend.names = NULL, curveColors = NULL, size = 22)
```

## Arguments

| | |
|---|---|
| `metaObject` | a metaObject which must have `metaObject$originalData` populated with a list of `datasetObjects` that will be used for discovery |
| `filterObject` | a metaFilter object containing the signature genes that will be used for calculating the score |
| `title` | title of the plot |
| `legend.names` | the name listed for each dataset in the legend (default: the `datasetObject$formattedName` for each dataset) |
| `curveColors` | *Graphical:* vector of colors for the PRC curves |
| `size` | use this to easily increase or decrease the size of all the text in the plot |

## Details

Each PRC plot evaluates the ability of a given gene set to separate two classes. As opposed to ROC curves, PRC curves are more sensitive to class imbalances. The gene set is evaluated as a Z-score of the difference in means between the positive genes and the negative genes (see calculateScore).

## Value

Returns a ggplot PRC plot for all datasets

## Author(s)

Aditya M. Rao, Andrew B. Liu

## See Also

`prcPlot`, `multipleROCPlot`

## Examples

```
multiplePRCPlot(tinyMetaObject, filterObject =
   tinyMetaObject$filterResults$pValueFDR0.05_es0_nStudies1_looaTRUE_hetero0)
```

## multipleROCPlot *Generate a plot with multiple ROC curves*

### Description

Generate a plot with multiple ROC curves

### Usage

```
multipleROCPlot(metaObject, filterObject, title = "title", size = 16)
```

### Arguments

| | |
|---|---|
| metaObject | a Meta object which must have the $originalData populated |
| filterObject | a MetaFilter object containing the signature genes that will be used for calculating the score |
| title | title of the plot |
| size | use this to easily increase or decrease the size of all the text in the plot |

### Value

Generates an ROC plot for all datasets

### Author(s)

Aditya M. Rao, Andrew B. Liu

### Examples

```
multipleROCPlot(tinyMetaObject, filterObject =
   tinyMetaObject$filterResults$pValueFDR0.05_es0_nStudies1_looaTRUE_hetero0)
```

## pooledROCPlot *Generate a plot with a pooled ROC curve*

### Description

Given a metaObject with $originalData populated, this function calculates and plots a "pooled" ROC curve that represents the average of all the individual ROC curves. This version of the function is for use with MetaIntegrator.

### Usage

```
pooledROCPlot(metaObject, filterObject, points = 1000,
  weighting = TRUE, title = NULL, size = 14, rounding = 3,
  smoothed = FALSE, auc1.thresh = 0.99, bootReps = 1000,
  minPoints = 5, numCores = 1, method = "random")
```

## Arguments

| | |
|---|---|
| metaObject | a metaObject which must have metaObject$originalData populated with a list of datasetObjects that will be used for discovery |
| filterObject | a metaFilter object containing the signature genes that will be used for calculating the score |
| points | number of points to simulate for the approximated ROC curves during the linear interpolation (default: 1000) |
| weighting | when calculating the mean AUC, if weighting=TRUE then the weighted mean AUC is calculated (default: TRUE) |
| title | title of the plot |
| size | size of the text/legend/etc (default: 14) |
| rounding | how many digits to round the AUC and CI to (default: 3) |
| smoothed | if TRUE, then a smoothed ROC curve is estimated using a modified version of the Kester and Buntinx Method |
| auc1.thresh | (if smoothed=TRUE) if the AUC of a dataset is above this threshold, then it is treated as if the AUC were 1 (default: 0.99) |
| bootReps | (if smoothed=TRUE) number of bootstrap iterations (default: 1000) |
| minPoints | (if smoothed=TRUE) minimum number of points required for bootstrap to be used (default: 5) |
| numCores | (if smoothed=TRUE) number of CPUs to use if parallel computing is desired (default: 1) |
| method | (if smoothed=TRUE) method used to compute summary meta-statistics (default: "random") |

## Details

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.

By default, this average ROC curve is calculated by first using linear interpolation to create approximated versions of each given ROC curve that all have the same set of FPR values. A pooled ROC curve is then calculated by taking the weighted mean of the corresponding TPR values (weighting corresponds to the number of samples in each dataset). This pooled curve is represented as a black curve. In addition, the weighted standard deviation is calculated for each TPR, which is represented by a grey area on the plot. The pooled AUC is calculated by using the trapezoid method on the pooled ROC curve, and the 95% confidence interval of the pooled AUC is calculated using the pooled standard error of the individual ROC curves.

If smoothed=TRUE, then a smoothed version of the pooled ROC curve will be plotted instead, with the surrounding gray area representing the weighted standard deviation of the pooled ROC curve. The statistics for this smoothed curve are based on the Kester and Buntinx Method, from (Kester and Buntinx, *Med Decis Making*, 2000). Methods have been added by Tim Sweeney (2015) for better estimates in cases with low numbers of tpr/fpr values. Methods have also been added by Aditya Rao (2018) to predict the curve's alpha parameter for a given beta parameter and AUC, as well as to calculate the weighted standard deviation of the given ROC curves.

## Value

Generates a plot with each individual ROC curve as well as the pooled ROC curve

## Author(s)

Aditya M. Rao (with help from Hayley Warsinske and Francesco Vallania, original idea from Madeleine Scott, and some code adapted from Tim Sweeney)

## References

Kester and Buntinx, *Med Decis Making*, 2000

## See Also

`summaryROCPlot`

## Examples

```
pooledROCPlot(tinyMetaObject, filterObject =
   tinyMetaObject$filterResults$pValueFDR0.05_es0_nStudies1_looaTRUE_hetero0)
```

---

# prcPlot *Plot the PRC Curve for a Dataset*

---

## Description

prcPlot will plot a Precision-Recall curve (and return the AUPRC) that describes how well a gene signature (as defined in a `filterObject`) classifies groups in a dataset (in the form of a `datasetObject`).

## Usage

```
prcPlot(filterObject, datasetObject, title = datasetObject$formattedName,
  subtitle = NULL, textSize = NULL, rounding = 3,
  curveColors = "red", legend = TRUE, PRC.lty = 1, PRC.lwd = 1,
  backgroundColor = "gray93", grid.marks = 0.1, grid.color = "white",
  grid.lty = 1, grid.lwd = 0.9, legend.lty = 0, cex.main = 1,
  cex.subtitle = 0.9)
```

## Arguments

| | |
|---|---|
| `filterObject` | a metaFilter object containing the signature genes that will be used for calculating the score |
| `datasetObject` | a Dataset object for group comparison in the PRC plot. (At least, must have a `$expr` of probe-level data, `$keys` of probe:gene mappings, and `$class` of two-class labels.) |
| `title` | title of the plot (default: `datasetObject$formattedName`) |

| | |
|---|---|
| `subtitle` | subtitle of the figure |
| `textSize` | use this to easily increase or decrease the size of all the text in the plot |
| `rounding` | how many digits to round the AUPRC and CI to (default: 3) |
| `curveColors` | *Graphical:* the color for the PRC curves (default: "red") |
| `legend` | *Graphical:* if TRUE, a legend will be included |
| `PRC.lty` | *Graphical:* PRC curve line type |
| `PRC.lwd` | *Graphical:* PRC curve line width |
| `backgroundColor` | *Graphical:* background color of the plot |
| `grid.marks` | *Graphical:* increment between grid lines |
| `grid.color` | *Graphical:* grid line color |
| `grid.lty` | *Graphical:* grid line type |
| `grid.lwd` | *Graphical:* grid line width |
| `legend.lty` | *Graphical:* legend style (0 is no box, 1 is boxed legend) |
| `cex.main` | *Graphical:* title size |
| `cex.subtitle` | *Graphical:* subtitle size |

## Details

Evaluates the ability of a given gene set to separate two classes. As opposed to ROC curves, PRC curves are more sensitive to class imbalances. The gene set is evaluated as a Z-score of the difference in means between the positive genes and the negative genes (see calculateScore).

## Value

Returns a standard PRC plot, plus AUPRC with 95% CI (calculated with the trapezoid method).

## Author(s)

Aditya M. Rao, Jiaying Toh

## See Also

`multiplePRCPlot`, `rocPlot`

## Examples

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

`predvalPlot` *Plot positive and negative predictive values across different prevalences*

## Description

Positive and negative predictive values (PPV and NPV) are two diagnostic statistics that change depending on the prevalence, so if you don't have a discrete prevalence to work with this function can create a plot that shows the positive and negative predictive values across all possible prevalences (as long as you have already calculated the sensitivity and specificity).

## Usage

```
predvalPlot(sens, spec, nsteps=1000, title=NULL, rounding=2)
```

## Arguments

| | |
|---|---|
| `sens` | the sensitivity of the prediction |
| `spec` | the specificity of the prediction |
| `nsteps` | the number of steps between prevalence 0% and 100% (i.e. the number of steps in the X-axis) (default: 1000) |
| `title` | title of the plot (if left blank, it will just indicate the input sensitivity and specificity) |
| `rounding` | number of significant digits for displaying the sensitivity, specificity, PPV, and NPV (default: 2) |

## Value

Plotly plot of predictive values vs. prevalence

## Author(s)

Lara Murphy, Aditya M. Rao

## Examples

```
predvalPlot(sens = 0.9, spec = 0.8)
```

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

## Description

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

## Usage

```
regressionPlot(filterObject, datasetObject,
  continuousVariableColumn = "continuous",
  formattedVariableName = "Continuous Variable", corMethod = "pearson",
  correlationCorner = "bottomRight")
```

## 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 specifies the continuous variable to compare (default: 'continuousVariableColumn') |
| `formattedVariableName` | label which will be used on the x-axis on the plot |
| `corMethod` | method which will be passed to cor.test |
| `correlationCorner` | one of topLeft, topRight, bottomLeft, bottomRight (default: bottomRight) |

## Value

Returns a regression plot as ggplot2 plot object

## Author(s)

Winston A. Haynes

## Examples

```
regressionPlot(tinyMetaObject$filterResults[[1]],
               tinyMetaObject$originalData$Whole.Blood.Study.1,
               continuousVariableColumn="age",
               formattedVariableName="Age")
```

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$leaveOneOutAna`

## 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) |

## Details

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.

## Value

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

## Author(s)

Francesco Vallania, Aditya M. Rao

## 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)
```

## subsetOriginalData *Subset samples for a particular dataset*

### Description

Subset samples for a particular dataset

### Usage

```
subsetOriginalData(datasetObject, keepMe)
```

### Arguments

| | |
|---|---|
| datasetObject | the Dataset object to subset |
| keepMe | either a binary vector for whether each sample should be in the subset or a list of names of samples to be in the subset |

### Details

Subsets all relevant slots within the Dataset object to include only the desired samples.

### Value

returns a Dataset object that has been subsetted to the desired samples

### Author(s)

Winston A. Haynes

### Examples

```
subsetObject <- subsetOriginalData(tinyMetaObject$originalData$Whole.Blood.Study.1,
   keepMe= c("Sample 1", "Sample 13", "Sample 43"))
```

## 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

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))
```

---

## `summaryROCCalc` *Calculate the summaryROC statistics*

---

### Description

Calculate the summaryROC statistics

### Usage

```
summaryROCCalc(metaObject, filterObject, bootstrapReps = 500)
```

### Arguments

| | |
|---|---|
| `metaObject` | a Meta object which must have the $originalData populated |
| `filterObject` | a MetaFilter object containing the signature genes that will be used for calculating the score |
| `bootstrapReps` | number of bootstrap simulations to run for confidence interval on summary ROC |

### Value

Summary AUC statistics

### Author(s)

Timothy E. Sweeney

### Examples

```
## Not run:
summaryROCCalc(tinyMetaObject, filterObject =
   tinyMetaObject$filterResults$pValueFDR0.05_es0_nStudies1_looaTRUE_hetero0)

## End(Not run)
```

---

## summaryROCPlot *Generate a plot with a summary ROC curve*

---

### Description

Generate a plot with a summary ROC curve

### Usage

```
summaryROCPlot(metaObject, filterObject, bootstrapReps = 500,
  orderByAUC = TRUE, alphaBetaPlots = TRUE)
```

### Arguments

| | |
|---|---|
| `metaObject` | a Meta object which must have the $originalData populated |
| `filterObject` | a MetaFilter object containing the signature genes that will be used for calculating the score |
| `bootstrapReps` | number of bootstrap simulations to run for confidence interval on summary ROC |
| `orderByAUC` | if TRUE, then order legend by summary AUC. Otherwise, use default ordering. |
| `alphaBetaPlots` | if TRUE, then draw forest plots of alpha and beta. If false, suppress plotting. |

### Value

Generates a ROC plot for all datasets

### Author(s)

Timothy E. Sweeney

## Examples

```
## Not run:
summaryROCPlot(tinyMetaObject,filterObject =
   tinyMetaObject$filterResults$pValueFDR0.05_es0_nStudies1_looaTRUE_hetero0)

## End(Not run)
```

---

`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",
  comparisonMethod = "wilcox.test", pairwiseComparisons = TRUE,
  autoLineBreak = TRUE)
```

## 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 specifies the groups to compare, typically case or control (default: 'label') |
| `comparisonMethod` | statistical test that will be used (default="wilcox.test"). Other options include "t.test". |
| `pairwiseComparisons` | if TRUE, perform pairwise statistical comparisons against the first factor level. If FALSE, perform global statistical comparisons (default: TRUE). |
| `autoLineBreak` | if TRUE, insert line breaks into labels on plots. If FALSE, don't insert line breaks (default: TRUE) |

## 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")
```