Package ‘genBart’

March 13, 2018

Type Package

Title Generate 'BART' File

Version 1.0.1

Description A set of functions to generate and format results from statistical analyses of a wide range of high throughput experiments that can then be uploaded into the 'BART' (Bio-Statistical Analysis Reporting Tool) 'shiny' app <https://github.com/jcardenas14/BART>. The app provides users with tools to visualize and efficiently sift through large amounts of data and results.

Depends R (>= 3.4.0)

Imports clValid (>= 0.6-6), data.table (>= 1.10.4), fastcluster (>= 1.1.22), ggplot2 (>= 2.2.1), grid (>= 3.3.3), gtools (>= 3.5.0), limma (>= 3.30.13), NMF (>= 0.20.6), pca3d (>= 0.10), psych (>= 1.7.3.21), qusage (>= 2.6.1), RColorBrewer (>= 1.1-2), reshape2 (>= 1.4.2), rmarkdown (>= 1.5), scales (>= 0.4.1), shiny (>= 1.0.3), shinydashboard (>= 0.5.3), shinyjs (>= 0.9), statmod (>= 1.2.2), stats (>= 3.3.3), stringr (>= 1.2.0), tools (>= 3.3.3), VennDiagram (>= 1.6.17)

Suggests DESeq2 (>= 1.14.1), edgeR (>= 3.16.5), knitr (>= 1.15.1), SummarizedExperiment (>= 1.1.6), testthat

License GPL-2 | GPL-3

Encoding UTF-8

LazyData true

LazyDataCompression xz

VignetteBuilder knitr

RoxygenNote 6.0.1

biocViews Microarray, RNASeq, AnnotationData

NeedsCompilation no

Author Jacob Cardenas [aut, cre],
Jacob Turner [aut],
Derek Blankenship [aut]

Maintainer Jacob Cardenas <jacob.cardenas@bswhealth.org>

Repository CRAN

Date/Publication 2018-03-13 22:13:14 UTC

R topics documented:

`clusterData` *Hierarchical clustering of normalized expression data*

Description

Perform hierarchical clustering on normalized data

Usage

```
clusterData(norm.data, dist.method = "euclidean", agg.method = "complete")
```

Arguments

`norm.data`	list of normalized expression data returned by `normalizeData`
`dist.method`	The distance measure to be used. This must be one of "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski". See `dist` for more details.
`agg.method`	The agglomeration method to be used. This must be one of "single", "complete", "average", "mcquitty", "ward.D", "ward.D2", "centroid" or "median". `hclust` for more details.

Details

This function performs hierarchical clustering on the rows of the normalized expression data contained in `norm.data`.

Value

`rowdend1b` dendrogram from hierarchical clustering of genes on baseline samples normalized according to `norm.method` specified in `norm.data`. NULL if `y1b` in `norm.data` is NULL.

`rowdend2b` dendrogram from hierarchical clustering of genes on baseline samples normalized to controls according to `norm.method` specified in `norm.data`. NULL if `y2b` in `norm.data` is NULL.

`rowdend1` dendrogram from hierarchical clustering of genes on all samples normalized according to `norm.method` specified in `norm.data`. NULL if `y1` in `norm.data` is NULL.

`rowdend2` dendrogram from hierarchical clustering of genes on all samples normalized to controls according to `norm.method` specified in `norm.data`. NULL if `y2` in `norm.data` is NULL.

`rowdend3` dendrogram from hierarchical clustering of genes on all samples normalized to their baseline. NULL if `y3` in `norm.data` is NULL.

`norm.method` string describing the normalization method used in `normalizeData`

Examples

```
# Example data
data(tb.expr)
data(tb.design)

# Use first 100 probes to demonstrate
dat <- tb.expr[1:100,]

# Create desInfo object
meta.data <- metaData(y = dat, design = tb.design, data.type = "microarray",
                    columnname = "columnname", long = TRUE, sample.id = "sample_id",
                    subject.id = "monkey_id", time.var = "timepoint",
                    baseline.var = "timepoint", baseline.val = 0)

# Normalize data
data.norm <- normalizeData(meta = meta.data)

# Cluster data
dendros <- clusterData(norm.data = data.norm)
```

`crossCorr` *Cross Correlations*

Description

Perform pairwise correlations formatted for BART

Usage

```
crossCorr(x, y, by = NULL, by.name = NULL, description = "X vs Y",
  x.var = "X", y.var = "Y", method = "pearson", order.by = "p",
  decreasing = TRUE)
```

Arguments

`x`	A numeric matrix or data frame.
`y`	A second numeric matrix or data frame with the same number of rows as `x`.
`by`	Default is NULL. A grouping vector with length equal to `nrow(x)`. Allows correlation by group (e.g. time).
`by.name`	Default is NULL. String denoting the name of the grouping factor.
`description`	String giving description of what's being correlated. Default is "X vs Y".
`x.var`	String denoting type of variables in `x`. Default is "X".
`y.var`	String denoting type of variables in `y`. Default is "Y".
`method`	Default is set to "pearson". The alternatives are "spearman" and "kendall".
`order.by`	Order by p-value ("p") or correlation value ("r"). Default is "p".
`decreasing`	logical. Should the sort be increasing or decreasing? Default is TRUE.

Details

This function uses the `corr.test` function in the pysch package to find the correlations and p-values. It then formats the results in a tall format for BART.

Value

`corrs` Tall data frame of correlations and p-values.

`corr.files` A data frame obtained by `cbind(by,x,y)`.

`corr.names` A string denoting the description of the variables being correlated.

`x.var` String denoting type of variables in `x`.

`y.var` String denoting type of variables in `y`.

`corr.method` One of "pearson", "spearman", or "kendall".

Examples

```
# Example data
data(tb.flow)

# Format expression data to align with flow data
gene.data <- data.frame(t(tb.expr[1:100, ]))
rownames(gene.data) <- paste0(tb.design$monkey_id, "_", tb.design$timepoint)
flow.data <- data.frame(t(tb.flow))
flow.data <- flow.data[match(rownames(gene.data), rownames(flow.data), nomatch = 0), ]
gene.data <- gene.data[match(rownames(flow.data), rownames(gene.data), nomatch = 0), ]

# Create time variable
time <- tb.flow.des$timepoint[match(rownames(flow.data),tb.flow.des$columnname,nomatch = 0)]

# Run correlations and format for BART
corrs <- crossCorr(x = gene.data, y = flow.data, by = time, by.name = "days",
                   description = "Genes vs Flow", x.var = "Genes",
                   y.var = "Flow", method = "spearman")
```

gene.symbols *Gene Symbols*

Description

A vector of gene symbols to match the probes ids in expression

Usage

```
gene.symbols
```

Format

A vector of 1000 gene symbols

Source

Illumina

genFile *Generate and Update BART Result files*

Description

Generate/Update BART file that can be directly uploaded into the BART app

Usage

```
genFile(meta, model.results = NULL, module.scores = NULL,
  dendrograms = NULL, qusage.results = NULL, roast.results = NULL,
  corr.results = NULL, project.name = "BART Project", folder.path = NULL)

updateFile(load.path = NULL, output.path = NULL, meta = NULL,
  model.results = NULL, module.scores = NULL, dendrograms = NULL,
  qusage.results = NULL, roast.results = NULL, corr.results = NULL,
  project.name = NULL)
```

Arguments

`meta`	A list of a single or multiple objects returned by `metaData`. Default is NULL.
`model.results`	A list of a single or multiple objects returned by `genModelResults`. Default is NULL.
`module.scores`	Object returned by `genModScores`. Default is NULL.
`dendrograms`	Object returned by `clusterData`. Default is NULL.
`qusage.results`	Object returned by `runQgen`. Default is NULL.

`roast.results`	Object returned by `rBart`. Default is NULL.
`corr.results`	A list of a single or multiple objects returned by `crossCorr`. Default is NULL.
`project.name`	String giving the name project name. Default is "BART Project".
`folder.path`	Path to create folder that BART file will be saved to. If NULL (default), BART folder will be created in the current working directory.
`load.path`	Folder path of BART file that needs to be updated.
`output.path`	Folder path to save updated BART file. If NULL (default), updated BART file will override the old one.

Details

`genFile` generates a formatted R data file (bartResults.rda) that can be uploaded into the BART web application. The file is created based on result objects returned by the various functions in `genBart`. Since BART can store and display results across multiple platforms at one time (e.g. RNA-seq, flow cytometry, metabolomics), some of the parameters (i.e. meta, model.results, corr.results) require the input objects to be wrapped in a list. The function saves the file in a folder whose name is given by `project.name`. The folder is created in the current working directory.

`updateFile` takes an existing BART file and allows the user to easily add and/or revise components. The user can override the old BART file by leaving `output.path` as NULL or give a new path in which to save the updated file.

Examples

```
# Example data
data(tb.expr)
data(tb.design)

# Use first 100 probes to demonstrate
dat <- tb.expr[1:100,]

# Create desInfo object
meta.data <- metaData(y = dat, design = tb.design, data.type = "microarray",
                      columnname = "columnname", long = TRUE, subject.id = "monkey_id",
                     baseline.var = "timepoint", baseline.val = 0, time.var = "timepoint",
                      sample.id = "sample_id")

# create BART file (minimal example)
genFile(meta = list(meta.data), folder.path = tempdir())

# generate module scores, normalize and cluster genes
mods <- genModScores(meta.data, modules)
data.norm <- normalizeData(meta = meta.data)
dendros <- clusterData(norm.data = data.norm)

# Update BART file with module scores and clustered genes
path <- paste0(tempdir(), "/", "BART Project")
updateFile(load.path = path, module.scores = mods, dendrograms = dendros)
```

genModelResults — Generate formatted results file

Description

Generate formatted results file from result objects returned by limma, DESeq2, and edgeR pipelines

Usage

```
genModelResults(y = NULL, data.type = "rnaseq", method = "limma", object,
  lm.Fit = NULL, comp.names = NULL, var.symbols = NULL,
  gene.sets = NULL, annotations = NULL)
```

Arguments

`y`	Expression data frame the model was run on. Default is NULL. See Details for further description.
`data.type`	Type of data being analyzed ("rnaseq", "microarray", "flow", "metab"). Default is data.type="rnaseq".
`method`	string denoting modeling method used ("limma","deseq2","edgeR")
`object`	results object generated from `eBayes` (limma), `results` (DESeq2), `glmLRT` or `glmQLFTest` (edgeR). Objects generated from `results`, `glmLRT`, or `glmlQLFTest` must be wrapped in a list. See Details for further description.
`lm.Fit`	linear model fit object generated from `lmFit` (limma). This parameter can can be left as NULL when `method` equals "deseq2" or "edgeR".
`comp.names`	Optional vector of comparison (contrast) names. Default is NULL.
`var.symbols`	Optional vector of additional annotations for the variables. Otherwise, the rownames of the expression data are used. Default is NULL.
`gene.sets`	A list of gene sets. Default is NULL. See Details for further description.
`annotations`	A data frame of additional annotations for the gene sets. Default is NULL. See Details for further description.

Details

This function takes results obtained from differential analysis pipelines found in `limma`, `DESeq2`, or `edgeR` and formats them for the BART app.

The expression data `y` and `lm.Fit` objects are used to obtain the residual matrix from the fitted model. These parameters are only needed when method = "limma" and data.type = "rnaseq" or "microarray" and can otherwise be left as NULL. It is important to remember that `y` should be the expression data used for modeling (e.g. voom transformed data). The residual matrix is stored as an element of the returned list and can be used in downstream gene set analysis using `runQgen` (Please visit for more details).

The `object` parameter takes as input model result objects returned by functions in limma, DESeq2, or edgeR. When method = "limma", the expected input is the single object returned by `eBayes` since it is able to store results across multiple comparisons. When method = "deseq2" or "edgeR", the result object(s) returned by results, glmLRT, or glmQLFTest must be wrapped in a list in which each element is an object containing the results for a single comparison.

The `comp.names` parameter is a character vector of comparison names that is particularly useful when method = "deseq2" or "edgeR" since comparison names are not extracted from the result objects generated by either of those pipelines. When using limma, the comparison names can also be defined in makeContrasts. It is important that the names are written in the correct order. For example, if object = list(AvsB, CvsD), where AvsB and CvsD are result objects for the comparisons "group A vs group B" and "group C vs group D" respectively, then comp.names = c("CvsD", "AvsB") would incorrectly assign the name "CvsD" to the comparison "group A vs group B" and vice versa. The `var.symbols` parameter is typically used to provide a character vector of gene symbols. The vector provided must be the same length and in the same order as the row names of the data used for modeling.

The `gene.sets` parameter is a list in which each element is a character vector of gene names comprising a gene set. The gene names must match the rownames of the data used for modeling. The gene sets are used to create modular maps for each comparison in the DGE section of BART. The `annotations` parameter is a data frame consisting of two columns. The first column consists of gene set names and the second column consists of additional descriptions for the gene sets.

Value

`data.type` string denoting the type of data that was analyzed

`results` the formatted results returned as a data frame

`resids` data frame of residuals. Returned only if data.type="microarray" or "rnaseq" and method="limma". Used to estimate the VIFs when running the Qusage algorithm in runQgen.

`gene.sets` list of gene sets provided by the user. NULL if no list provided.

`annotations` data frame of gene set annotations provided by the user. Null if no annotations are provided.

Examples

```
# Example data
data(tb.expr)
data(tb.design)

# Only use first 100 genes to demonstrate
dat <- tb.expr[1:100,]

# Generate lmFit and eBayes (limma) objects needed for genModelResults
tb.design$Group <- paste(tb.design$clinical_status,tb.design$timepoint,sep = "")
grp <- factor(tb.design$Group)
design2 <- model.matrix(~0+grp)
colnames(design2) <- levels(grp)

dupcor <- limma::duplicateCorrelation(dat, design2, block = tb.design$monkey_id)
fit <- limma::lmFit(dat, design2, block = tb.design$monkey_id,
             correlation = dupcor$consensus.correlation)
contrasts <- limma::makeContrasts(A_20vsPre = Active20-Active0, A_42vsPre = Active42-Active0,
                                  levels=design2)
fit2 <- limma::contrasts.fit(fit, contrasts)
fit2 <- limma::eBayes(fit2, trend = FALSE)

# Format results
model.results <- genModelResults(y = dat, data.type = "microarray", object = fit2,
                                 lm.Fit = fit, method = "limma")
```

---

genModScores   *Generate modular (gene set) maps for plotting*

Description

Generate modular (gene set) maps for plotting

Usage

```
genModScores(meta, gene.sets, sd.lim = 2, annotations = NULL)
```

Arguments

`meta`	list returned by `metaData`
`gene.sets`	list of gene sets. See `genModelResults` for additional formatting details.
`sd.lim`	number of standard deviations away from the mean of the reference samples. Default is 2.
`annotations`	A data frame of additional annotations for the gene sets. Default is NULL. See `genModelResults` for additional details.

Details

This function calculates module scores for individual samples. In cross sectional studies with controls, the control samples are used to determine an upper and lower threshold (mean of controls +/- 2 sd). The module proportion for each sample is then calculated based on the percentage of genes within a module that are above or below this threshold. For example, if 40% of the genes within a module are above the threshold and 15% are below it, the final module score would be 25% up (40-15). In longitudinal settings, module scores are calculated with respect to controls and baseline samples. In cross sectional studies without controls, `genModules` cannot be used, since there are no reference samples with which to calculate a threshold.

Value

`scores.ctrl` data frame of module scores for all samples with respect to controls.

`scores.base` data frame of module scores for all time point samples with respect to their baseline.

`gene.sets` List of gene sets provided through `gene.sets`

Examples

```
# Example data
data(tb.expr)
data(tb.design)
data(modules)

# Demonstrate on first 100 probes
dat <- tb.expr[1:100, ]

# Create desInfo object
meta.data <- metaData(y = tb.expr, design = tb.design, data.type = "microarray",
                    columnname = "columnname", long = TRUE, sample.id = "sample_id",
                    subject.id = "monkey_id", time.var = "timepoint",
                    baseline.var = "timepoint", baseline.val = 0)

# Generate module maps
mods <- genModScores(meta.data, modules)
```

---

metaData *Declare meta data information for downstream analysis*

Description

Match design and expression data frames. Declare and store design parameters.

Usage

```
metaData(y, design, data.type = "rnaseq", columnname = NULL, long = FALSE,
  time.var = NULL, sample.id = NULL, subject.id = NULL,
  baseline.var = NULL, baseline.val = NULL, control.var = NULL,
  control.val = NULL)
```

Arguments

`y`	An expression data frame.
`design`	A sample annotation data frame containing sample information (e.g. age, condition, timepoint, etc.).
`data.type`	Type of data that's going to be analyzed ("rnaseq", "microarray","flow", "metab"). Default is data.type="rnaseq".
`columnname`	Name of column in design that contains the column names of y.
`long`	logical; Is the study longitudinal?.
`time.var`	For longitudinal studies. Name of column in design that contains the study time points.
`sample.id`	Name of column in design that contains unique sample identification.
`subject.id`	Name of column in design that contains ids for individual subjects.

`baseline.var`	Name of column in design that contains values referring to baseline observations.
`baseline.val`	String or numeric value denoting baseline observations.
`control.var`	Name of column in design that contains values referring to controls.
`control.val`	String or numeric value denoting controls.

Value

A list containing the matched design and expression data and all of the design parameters specified.

Examples

```
# Using example data
data(tb.expr)
data(tb.design)
meta.data <- metaData(y = tb.expr, design = tb.design, data.type = "microarray",
                    columnname = "columnname", long = TRUE, sample.id = "sample_id",
                    subject.id = "monkey_id", time.var = "timepoint",
                    baseline.var = "timepoint", baseline.val = 0)
```

`modules` *Baylor Modules*

Description

A list containing the baylor modules (gene sets)

Usage

```
modules
```

Format

A large list containing 260 gene sets.

Source

baylor modules

`normalizeData` *Data Normalization*

Description

Perform various normalizations of expression data

Usage

```
normalizeData(meta, norm.method = "mean")
```

Arguments

`meta`	list returned by `metaData`
`norm.method`	String denoting whether to normalize to the mean or median of all samples or a control group specified in `meta`. Default is norm.method = "mean".

Details

This function performs various normalizations of the expression data, depending on the study design and the parameters defined in `metaData`. For all study designs, the data is normalized to the mean (or median) of all the samples. For cross-sectional studies with controls, an additional normalization to the mean (or median) of the controls is performed. For longitudinal designs, baseline normalization (subtract out each subject's baseline) and normalization to the mean (or median) of controls (if present) is performed. In addition, separate normalizations on baseline samples is performed.

Value

`y1b` data frame of baseline samples normalized according to `norm.method`. NULL if baseline samples are not specified in `meta`.

`y2b` data frame of baseline samples normalized to controls according to `norm.method`. NULL if control samples are not specified in `meta`.

`y1` data frame of all samples normalized according to `norm.method`.

`y2` data frame of all samples normalized to controls according to `norm.method`. NULL if control samples are not specified in `meta`.

`y3` data frame of all samples normalized to their baseline. NULL if study is not longitudinal or if baseline samples are not specified in `meta`.

`norm.method` string describing normalization method used.

Examples

```
# Example data
data(tb.expr)
data(tb.design)

# Use first 100 probes to demonstrate
dat <- tb.expr[1:100,]

# Create desInfo object
meta.data <- metaData(y = dat, design = tb.design, data.type = "microarray",
                    columnname = "columnname", long = TRUE, sample.id = "sample_id",
                    subject.id = "monkey_id", time.var = "timepoint",
                    baseline.var = "timepoint", baseline.val = 0)

# Normalize and cluster data
data.norm <- normalizeData(meta = meta.data)
```

`runBart` *Run BART Shiny App*

Description

This function runs the BART shiny app

Usage

```
runBart()
```

Examples

```
## Only run this example in interactive R sessions
if (interactive()) {
  runBart()
}
```

`runQgen` *Run Q-Gen (generalized QuSAGE) algorithm using gene level statistics*

Description

Run Q-Gen (generalized QuSAGE) algorithm using gene level statistics

Usage

```
runQgen(model.results, gene.sets, annotations = NULL)
```

Arguments

`model.results`	object returned by `genModelResults`.
`gene.sets`	list of gene sets. See `genModelResults` for more formatting details.
`annotations`	A data frame of additional annotations for the gene sets. See `genModelResults` for more formatting details.

Details

This function takes the gene level comparison estimates and test statistics contained in the object returned by `genModelResults` and runs the Q-Gen algorithm across all of the comparisons. The VIFs are estimated using the raw residuals, which are also contained in the output of `genModelResults`.

Value

`qusage.results` Tall formatted matrix of results

`lower.ci` Matrix of gene level lower 95% confidence intervals

`upper.ci` Matrix gene level upper 95% confidence intervals

`gene.sets` List of gene sets provided to `gene.sets`

`annotations` data frame of gene set annotations. Default is NULL.

Examples

```
# Example data
data(tb.expr)
data(tb.design)

# Use first 100 probes to demonstrate
dat <- tb.expr[1:100,]

# Create desInfo object
meta.data <- metaData(y = dat, design = tb.design, data.type = "microarray",
                    columnname = "columnname", long = TRUE, subject.id = "monkey_id",
                   baseline.var = "timepoint", baseline.val = 0, time.var = "timepoint",
                    sample.id = "sample_id")

# Generate lmFit and eBayes (limma) objects needed for genModelResults
tb.design$Group <- paste(tb.design$clinical_status,tb.design$timepoint, sep = "")
grp <- factor(tb.design$Group)
design2 <- model.matrix(~0+grp)
colnames(design2) <- levels(grp)
dupcor <- limma::duplicateCorrelation(dat, design2, block = tb.design$monkey_id)
fit <- limma::lmFit(dat, design2, block = tb.design$monkey_id,
                    correlation = dupcor$consensus.correlation)
contrasts <- limma::makeContrasts(A_20vsPre = Active20-Active0, A_42vsPre = Active42-Active0,
                                  levels=design2)
fit2 <- limma::contrasts.fit(fit, contrasts)
fit2 <- limma::eBayes(fit2, trend = FALSE)

# Create model results object for runQgen
```

```
model.results <- genModelResults(y = dat, data.type = "microarray", object = fit2, lm.Fit = fit,
                                 method = "limma")

# Run Q-Gen on baylor modules
data(modules)
qus.results <- runQgen(model.results, modules)
```

`tb.design` *Microarray design file*

Description

A microarray design file dataset for cynomolgus macaques infected with M. tuberculosis. Contains sample information such as time point and clinical status.

Usage

```
tb.design
```

Format

A data frame with 30 rows and 7 variables.

columnname names matching column names in expression dataset

monkey_id id assigned to annotate each monkey

timepoint times at which samples were drawn, in days

timerange groups time points into stages

sample_id id assigned to annotate each sample

clinical_status denotes if disease is active or latent

Source

http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE84152

`tb.expr` *Microarray data on cynomolgus macaques infected with M. tuberculosis*

Description

A microarray dataset on cynomolgus macaques infected with M. tuberculosis. The study was conducted on 38 monkeys at 11 unequally spaced time points. The dataset presented is a subset of the full dataset.

Usage

```
tb.expr
```

Format

A data frame with 4000 rows and 30 columns, where each row is a probe and each column a sample.

Source

http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE84152

`tb.flow` *Flow data on cynomolgus macaques infected with M. tuberculosis*

Description

A dataset containing flow variables of absolute counts and percentages.

Usage

```
tb.flow
```

Format

A data frame with 13 rows of flow variables and 213 columns of samples

Source

http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE84152

`tb.flow.des` *Flow design file*

Description

A flow design file dataset for cynomolgus macaques infected with M. tuberculosis. Contains sample information such as time point and clinical status.

Usage

```
tb.flow.des
```

Format

A data frame with 213 rows of samples and 7 variables.

columnname names matching column names in expression dataset

monkey_id id assigned to annotate each monkey

timepoint times at which samples were drawn, in days

timerange groups time points into stages

sample_id id assigned to annotate each sample

clinical_status denotes if disease is active or latent

Source

http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE84152

Index