Package 'ThreeDRNAseq'

November 27, 2018

Type Package

Title 3D RNA-seq App enables RNA-seq data analysis to be done only within 3 Days.

Version 0.1.0

Author Wenbin Guo <wenbin.guo@hutton.ac.uk>

Maintainer Wenbin Guo <wenbin.guo@hutton.ac.uk>

Description This package provides an interactive graphical user interface (GUI) for RNA-seq data differential expression (DE), differential alternative splicing (DAS) and differential transcript usage (DTU) analyses based on two popular pipelines: limma and edgeR. The 3D RNA-seq GUI is based on R shiny App and enables command-line-free analysis. To perform analysis, the first step is to generate transcript quantification from quantification tools, such as Salmon and Kallisto (1-2 Days). Then users can do mouse click on the App to upload transcript read counts, perform DE and DAS analysis, and make beautiful plots, e.g. expression mean-variance trend plots, PCA plots, heatmap, GO annotation plots, etc. (1 Day or less). The pipeline has steps of proper data pre-processing and false positive controls. These lead to robust 3D predictions. All the results and plots can be saved to a report in html, pdf or word format. The analysis pipeline has been successfully used in different RNA-seq studies from Arabidopsis, barley and potato.

License GPL-3

Encoding UTF-8

LazyData true

Depends R (>= 3.5.0)

Imports txiimport,
limma,
edgeR,
RUVSeq,
ggplot2,
ComplexHeatmap,
shiny,
shinydashboard,
shinyFiles,
plotly,
eulerr,
gridExtra,
Gmisc

RoxygenNote 6.1.1

Suggests BiocStyle, knitr, rmarkdown

VignetteBuilder knitr

R topics documented:

`boxplot.normalised` *Boxplot of expression distribution in each sample.*

Description

Boxplot of expression distribution in each sample.

Usage

```
## S3 method for class 'normalised'
boxplot(data.before, data.after, condition,
  sample.name)
```

Arguments

data.before	a numeric matrix of expression before normalisation, e.g. raw read counts.
data.after	a numeric matrix of expression after normalisation, e.g. $\log_2$-CPM.
condition	a vector of conditions, which is used to distinguish and colour the conditions of samples.
sample.name	a vector of sample names, which is used to name and order the samples in the boxplot.

Value

a list with two elements: "g1" and "g2", which are ggplot format boxplots of the data before and after normalisation.

---

check.mean.variance *Compare expression mean-variance trends before and after low expression filters.*

Description

Compare expression mean-variance trends before and after low expression filters.

Usage

```
check.mean.variance(counts.raw, counts.filtered, condition, span = 0.5,
  ...)
```

Arguments

counts.raw	a matrix/data.frame of raw read counts before low expression filters.
counts.filtered	a matrix/data.frame of read counts after low expression filters.
condition	a vector of characters to distinguish conditions of samples (e.g. c('A','A','B','B')), which is used to make the design matrix to fit linear regression model.
span	a numeric value passed to lowess smoothing window as a proportion to fit the mean-variance trend.
...	additional arguments passed to trend.mean.variance function.

Value

a list object with elements "fit.raw" and "fit.flitered", which are the trend.mean.variance fit results for counts.raw and counts.filtered, respectively.

See Also

trend.mean.variance and voom.

condition2design *Convert conditions to design matrix for linear regression*

Description

Convert conditions to design matrix for linear regression

Usage

```
condition2design(condition, batch.effect = NULL)
```

Arguments

batch.effect	batch effect terms estimated from remove.batch. Default is NULL.
condtion	a vector of conditions of samples corresponding to the expression dataset.

Value

A design matrix for linear regression.

counts2CPM *Convert read counts to counts per million reads (CPM)*

Description

Convert read counts to counts per million reads (CPM)

Usage

```
counts2CPM(obj, lib.size = NULL, Log = F)
```

Arguments

obj	a read count matrix or a DGEList object.
lib.size	a numeric vector of library size (sequencing depth) of each sample. If NULL, library size of a sample is calculated from sum of all reads in this sample.
Log	logcial, whether to take $\log_2$ of CPM.

Details

CPM is defined as:

$$\frac{counts}{lib.size} \times 10^6$$

To avoid taking $\log_2$ of zero values, small offsets 0.5 and 1 were added to the numerator and denominator, respectively, i.e.

$$\log_2 \frac{counts + 0.5}{lib.size + 1} \times 10^6$$

Value

a numeric matrix of CPM or $\log_2$-CPM.

See Also

cpm.

Examples

```
y <- matrix(rnbinom(20,size=1,mu=10),5,4)
counts2CPM(y,Log = T)
counts2CPM(y,Log = F)
```

`CPM.filter` *Filter low expression based on CPM (count per million reads)*

Description

Filter low expression based on CPM (count per million reads)

Usage

```
CPM.filter(counts, sample.n = 3, cpm.cut = 1, ...)
```

Arguments

`counts`	a matrix/data.frame or matrix of read counts.
`sample.n`	number of samples
`cpm.cut`	a numeric value of CPM cut-off

Details

An expressed target must have $\geq$ `sample.n` with CPM $\geq$ `cpm.cut`.

Value

A vector of logcial values (TRUE/FALSE) to indicate the rows of the input data to keep/filter.

Examples

```
x <- rbind(matrix(rpois(50,lambda=4),5,10),matrix(rpois(50,lambda=0.1),5,10))
rownames(x) <- letters[1:10]
filter.idx <- CPM.filter(x,sample.n = 3,cpm.cut = 1)
##The remaining
x[filter.idx,]
rownames(x)[filter.idx]
```

data.error *Calculate standard deviation or standard errors*

Description

Calculate standard deviation or standard errors from a numerical vector.

Usage

```
data.error(x, error.type = c("stderr", "sd"))
```

Arguments

x	a numerical vector
error.type	method used to calculate data error. Options include "sd" for standard deviation and "stderr" for standard error.

Value

data error

See Also

sd

Examples

```
set.seed(2000)
x=rnorm(1000,mean=0,sd=1)
##standard error
data.error(x,error.type = 'stderr')
##standard deviation
data.error(x,error.type = 'sd')
```

DEvsDAS *Compare DE and DAS results*

Description

Compare DE and DAS results

Usage

```
DEvsDAS(DE_genes, DAS_genes, contrast)

DEvsDTU(DE_trans, DTU_trans, contrast)

summary3Dnumber(DE_genes, DAS_genes, DE_trans, DTU_trans, contrast)
```

Arguments

DE_genes, DAS_genes, DE_trans, DTU_trans	data.frame objects of DE genes, DAS genes, DE transcripts and DTU transcripts, which are the outputs of summaryDEtarget and summaryDAStarget.
contrast	a vector of contrast groups, e.g. contrast = c('B-A','C-A'), which compares condition B and C to condition A.

Details

In each contrast group, DEvsDAS compares the DE and DAS genes; DEvsDTU compares the DE and DTU transcripts; and summary3Dnumber summarises the DE/DAS/DTU gene/transcript numbers.

Value

a data.frame with first column of contrast goups, second column of DE only genes/transcripts, third column of DE&DAS genes or DE&DTU transcripts and fourth column of DAS only genes or DTU only transcripts.

---

distinct.color *Generate distinct colours*

Description

Generate distinct colours

Usage

```
distinct.color(n)
```

Arguments

n	number of colours to generate.

Value

a vector of n colours.

---

generate.report *Generate html, pdf and word report from Rmd (RMardkown) file*

Description

Generate html, pdf and word report from Rmd (RMardkown) file

Usage

```
generate.report(input.Rmd, report.folder, type = c("all",
  "html_document", "pdf_document", "word_document"))
```

Arguments

input.Rmd	the Rmd file used to generate the report.
report.folder	the folder to save the reports.
type	the ouput document type. Options are "all", "html_document", "pdf_document" and "word_document".

Value

the generated html, pdf or word report will be saved into the provided report.folder.

---

gg.color.hue *Generate HCL colours, which are the same to* ggplot *default colour palette*

Description

Generate HCL colours, which are the same to ggplot default colour palette

Usage

```
gg.color.hue(n)
```

Arguments

n	number of colours to generate.

Value

a vector of n colours.

---

limma.pipeline *Dfferential expression (DE) and alternative splicing (DAS) analysis*

Description

Two pipelines are provided to perform differential expression (DE), differential alternative splicing (DAS) gene and differential transcript usage (DTU) transcript analysis.

Usage

```
limma.pipeline(dge, contrast, span = 0.5, design, deltaPS = NULL,
  diffAS = F, adjust.method = "BH")

edgeR.pipeline(dge, method = c("glm", "glmQL"), design, contrast,
  diffAS = F, deltaPS = NULL, adjust.method = "BH")
```

Arguments

dge	a numeric matrix of read counts or a DGEList object. Counts must be non-negative and NAs are not permitted.
contrast	a vector of contrast groups for expression comparisons, e.g. c('B-A','C-A') compares conditions "B" and "C" to condition "A".
span	width of the lowess smoothing window as a proportion, which is passed to the voom function to estimate expression mean-variance trend.
design	design matrix with rows of samples and columns of conditions to fit a linear model.
deltaPS	a matrix of ΔPS values, with rows of transcripts and columns of contrast groups.
diffAS	logical, whether to perform DAS analysis. If TRUE, the diffSplice (limma pipeline) or diffSpliceDGE (edgeR pipeline) function is used to peform DAS gene and DTU transcript analysis.
adjust.method	method the adjust p-values for multiple comparisons. Default is "BH" and full details can be found in p.adjust.
method	method to use in the edgeR.pipeline. Options are "glm" or "glmQL".

Details

Abbreviation

- DE: differential expressed genes/transcripts.
- DAS: differential alternative spliced genes.
- DTU: differential transcript usage transcripts.

Statistics

- Contrast groups: refer to the conditions to compare, e.g. "B-A" and "C-A" compares the conditions "B" and "C" to condition "A".
- Adjusted p-value: in the expression comparitive analysis, each gene/transcript is fitted with a statistical model, reporting a p-value. However the testing is performed over a substantial number of genes/transcripts and each testing has a probability of making errors. Hence, all p-values in the analsysis are adjusted by controlling the false discovery rate (FDR). Please see the R function p.adjust for details.
- L_2FC: $\log_2$ fold change of expression based on contrast groups.
- ΔPS: Transcript level PS (percent of splice) is defined as the ratio of transcript average abundance of conditions divided by the average gene abundance. ΔPS is the PS differences of conditions based on the contrast groups. The abundance to calculate PS values can be TPMs or read counts.

DE gene/transcript analysis

A gene/transcript is identified as DE in a contrast group if L_2FC of expression $\geq$ a cut-off and with adjusted p-value < a cut-off.

Two pipelines are provided for DE analysis:

"limma-voom" and "edgeR". The "edgeR" pipeline includes two methods: "glmQL" (Genewise Negative Binomial Generalized Linear Models with Quasi-likelihood Tests) and "glm" (Genewise Negative Binomial Generalized Linear Models). In limma pipeline, L2FCs are calculated based count per million reads (CPMs) while in edgeR, L2FCs are based on read counts. From several real RNA-seq data analyses, high consensus is achieved between "limma-voom" and "glmQL" (>90

DAS gene and DTU transcript analysis

To test differential expression, the gene level expression is compared to transcript level expression in the contrast groups. A gene is DAS in a contrast group if adjusted p-value < cut-off and at least one transcript of the gene with ΔPS $\geq$ a cut-off. A transcript is DTU if adjusted p-value < a cut-off and ΔPS $\geq$ a cut-off.

Two pipelines for AS analysis:

- limma pipeline: To identify DAS genes, the expression of each transcript is compared to the weighted average expression of all the transcripts for the same gene (weight on transcript expression variance; the average expression can be treated as gene level expression). P-value of each test is converted to genewise p-value by using F-test or Simes method. To identify DTU transcript, each transcript is compared to the weighted average of all the other transcripts for the same gene. See the `diffSplice` function in `limma` package for details.
- edgeR pipeline: To identify DTU transcripts, log-fold-change of each transcript is compared to log-fold-change of entire gene (sum of all transcripts). The DTU transcript test p-values are summarised to genewise p-value by using F-test or Simes method to report DAS genes. See `diffSpliceDGE` function in `edgeR` R package for details.

Value

a list object with elements:

- `voom.object`: the results of `voom` function to estimate expression mean-variance trend.
- `fit.lmFit`: the results of `lmFit` to fit a linear regression model on conditions.
- `fit.glm`: the results of `glmFit` in the edgeR pipeline.
- `fit.glmQL`: the results of `glmQLFit` in the edgeR pipeline.
- `fit.contrast`: the results of `contrasts.fit`, i.e. incorporate the contrast groups to the linear regression model.
- `fit.eBayes`: the results of `eBayes`.
- `DE.pval.list`: a list object of which each element is the result of applying `topTable` or `topTags` to individual contrast groups.
- `DE.pval`: a data.frame object of adjusted p-values, with rows of genes/transcripts and columns of contrast groups.
- `DE.lfc`: a data.frame object of L_2FC, with rows of genes/transcripts and columns of contrast groups.
- `DE.stat`: a data.frame object with first column of target (genes/transcripts), second column of contrast (contrast groups), third column of adj.pval and fourth column of log2FC.
- `DE.stat.overalltest`: a data.frame object of testing statistics over all contrast groups rather than testing in individual contrast groups.

If alternative splicing analysis is performed (`diffAS=TRUE`), more results are returned:

- `fit.splice`: the results of `diffSplice` or `diffSpliceDGE` for individual contrast groups.
- `DTU.pval.list`: a list object of which each element is the DTU transcript result of `topSplice` or `topSpliceDGE` for individual contrast groups.
- `DTU.pval`: a data.frame object of adjusted p-values, with rows of transcripts and columns of contrast groups.
- `DTU.deltaPS`: a data.frame object of ΔPS, with rows of transcripts and columns of contrast groups.

- `DTU.stat:` a data.frame object with first column of target (transcripts), second column of contrast groups, third column of adj.pval and fourth column of deltaPS.
- `maxdeltaPS:` a data.frame object with rows of genes and columns of maximum ΔPS of the transcripts in the same gene in different contrast groups.
- `DAS.pval.F.list:` a list object of which each element is the DAS gene result of "F-test" using `topSplice` or `topSpliceDGE` in individual contrast groups.
- `DAS.pval.F:` a data.frame object of adjusted p-value with rows of genes and columns of contrast groups.
- `DAS.F.stat:` the DAS gene statistics using "F-test".
- `DAS.pval.simes.list:` a list object of "Simes-test" using `topSplice` or `topSpliceDGE` in individual contrast groups.
- `DAS.pval.simes:` a data.frame object of DAS gene adjusted p-values using "Simes" method.
- `DAS.simes.stat:` the DAS gene statistics using "Simes-test".

See Also

R packages: `limma` and `edgeR`.

`low.expression.filter` *Filter low expressed transcripts and genes*

Description

Filter low expressed transcripts and genes

Usage

```
low.expression.filter(abundance, mapping, abundance.cut = 1,
  sample.n = 3, Log = F, unit = c("counts", "TPM"))
```

Arguments

`abundance`	a data.frame/matrix of read counts/TPMs at transcript level.
`mapping`	a data.frame of transcript-gene name mapping, with first column of transcript list and second column of gene list.
`abundance.cut`	a numeric value of abundance cut-off.
`sample.n`	a number of samples cut across all samples.
`Log`	logical. If TRUE, the log2-CPM is used for filters when read counts are used as abundance.
`unit`	the abundance used to filter the low expressed genes/transcripts. Options are "counts" and "TPM".

Details

If read counts are used, they are converted to CPM (count per million reads) to filter low expressed transcripts.

1) An expressed transcript must have $\geq$ `sample.n` with CPM $\geq$ `cpm.cut`.

2) An expressed gene must have at least one expressed transcript.

Value

A list of following elements:

- trans_high: the expressed transcripts
- genes_high: the expressed genes
- mapping: the remaining mapping data.frame after filter the low expressed transcripts

---

`plotAbundance` *Plot gene and transcript expression profiles*

Description

Plot gene and transcript expression profiles

Usage

```
plotAbundance(data.exp, gene, mapping, genes.ann = NULL,
  trans.ann = NULL, trans.expressed = NULL, reps,
  x.lab = "Conditions", y.lab = "TPM", marker.size = 3,
  legend.text.size = 11, error.type = "stderr", error.bar = T,
  show.annotation = T, plot.gene = T)

plotPS(data.exp, gene, mapping, genes.ann = NULL, trans.ann = NULL,
  trans.expressed = NULL, reps, y.lab = "TPM", x.lab = "Conditions",
  marker.size = 3, legend.text.size = 11, show.annotation = T)
```

Arguments

`data.exp`	a data.frame of transcript level expression, e.g. read counts and TPM.
`gene`	a gene name to plot.
`mapping`	a data.frame of transcript-gene mapping (first column is transcript list and second column is gene list).
`genes.ann`	a data.frame of gene annotation with first column "target" and second column "annotation". Default is `NULL`.
`trans.ann`	a data.frame of transcript annotation with first column "target" and second column "annotation". Default is `NULL`.
`trans.expressed`	a vector of expressed transcripts. If not `NULL`, only the profiles of provided transcripts will be shown in the plot.
`reps`	a vecotr of replicate labels, which provide the grouping information to calculate the average expression in each conditions.
`x.lab, y.lab`	characters for x-axis and y-axis labels, respectively.
`marker.size`	a number passed to `geom_point` to control the point size in the plot.
`legend.text.size`	a number to control the legend text size.
`error.type`	error type to make the error bars. Options are: "stderr" for standard error and "sd" for standard deviation.

error.bar	logical, whether to show error bars on the profile plot.
show.annotation	logical, whether to show the annotations provided in genes.ann and trans.ann on the plot.
plot.gene	logical, whether to show the gene level expression on the plot. The gene expression is the total of all the transcript expression.

Details

plotAbundance is used to plot the gene and/or trnascript level abundance while plotPS is used to plot the percent spliced (PS) of transcript. PS is defined as the ratio of transcript expression to the total of all transcript expression (i.e. gene expression) in the same gene.

Value

a plot in ggplot format.

See Also

data.error

Examples

```
data(exp.data)
plotAbundance(data.exp = exp.data$trans_TPM,
              gene = 'AT1G01020',
              mapping = exp.data$mapping,
              genes.ann = exp.data$genes.ann,
              trans.ann = exp.data$trans.ann,
              reps = exp.data$samples$condition)

plotPS(data.exp = exp.data$trans_TPM,
              gene = 'AT1G01020',
              mapping = exp.data$mapping,
              genes.ann = exp.data$genes.ann,
              trans.ann = exp.data$trans.ann,
              reps = exp.data$samples$condition)
```

---

plotEulerDiagram *Plot Euler diagram*

Description

Plot Euler diagram

Usage

```
plotEulerDiagram(x, fill = gg.color.hue(length(x)),
  shape = c("ellipse", "circle"), ...)
```

Arguments

x	(1) a list of individuals in different sets for comparisons or (2) a vecotr of numbers of set relations.
fill	a vector of colours to fill the diagram.
shape	geometric shape used in the diagram.
...	arguments passed down to the eulerr::euler function.

Value

a Euler diagram

See Also

eulerr::euler.

Examples

```
x <- letters[1:10]
y <- letters[5:15]
z <- set2(x,y)
combo <- c(x=length(z$x.only),y=length(z$y.only),"x&y"=length(z$xy))
plotEulerDiagram(list(x=x,y=y))
plotEulerDiagram(combo)
```

---

plotFlowChart *Plot flow-chart graph of DE and DAS results*

Description

The flow-chart shows the results of DE vs DAS genes or DE vs DTU transcripts. For example, the expressed genes are divided into DE genes and not DE genes; the DE genes are divided into DE only (only transcription regulation) and DE&DAS (both transcription and AS regulation) genes; the not DE genes are divided into DAS only (only AS regulation) and no expression/AS change (no regulation). The same divisions to DE and/or DTU transcripts.

Usage

```
plotFlowChart(expressed, x, y, type = c("genes", "transcripts"),
  pval.cutoff = 0.01, lfc.cutoff = 1, deltaPS.cutoff = 0.1)
```

Arguments

expressed	a vector of expressed genes/transcripts.
x	a vector of DE genes/transcripts.
y	a vector of DAS genes/DTU transcripts.
type	a character to indicate the provided list type. Options are "genes" and "transcripts".
pval.cutoff, lfc.cutoff, deltaPS.cutoff	the cut-offs used to determine the significant genes/transcripts.

Value

a flow-chart plot.

`plotGO` *Bar plot of GO annotation*

Description

Bar plot of GO annotation

Usage

```
plotGO(go.table, col.idx, plot.title = "GO annotation")
```

Arguments

`go.table`	a data.frame of GO annotation table, with first column "Category" (i.e. BP, CC and MF), second column of Go "Term" and remaining columns with significant statistics, e.g. "Count", "-log10(FDR)", etc.
`col.idx`	a character to indicate which column of statistics to use to report the significance, e.g. `col.idx="-log10(FDR)"`. `col.idx` must match to the column names of `go.table`.
`plot.title`	the titile to show on the plot.

Value

a plot in `ggplot` format.

Examples

```
data(go.table)
plotGO(go.table = go.table,col.idx = '-log10(FDR)',plot.title = 'GO annotation: DE genes')
```

`plotMeanVariance` *Plot the mean-variance trend*

Description

Plot the mean-variance trend

Usage

```
plotMeanVariance(x, y, l, fit.line.col = "red",
  x.lab = "log2( count size + 0.5 )",
  y.lab = "Sqrt(standard deviation)", main = "Mean-variance trend",
  lwd = 1.5, ...)
```

Arguments

x	a numeric vector of mean value from the results of trend.mean.variance.
y	a numeric vector of variance value from the results of trend.mean.variance.
x.lab, y.lab	a string of x-axis label and y-axis label, respectively.
main	plot title.
...	additional arguments passed to plot.

Value

a plot

---

plotPCAind *Make principal component analysis (PCA) plot of individual samples*

Description

Make principal component analysis (PCA) plot of individual samples

Usage

```
plotPCAind(data2pca, dim1 = "PC1", dim2 = "PC2", groups,
  plot.title = "PCA plot", ellipse.type = c("none", "ellipse",
  "polygon"), add.label = T, adj.label = F)
```

Arguments

data2pca	a matrix/data.frame to make the PCA plot of which the rows are defined individuals to plot.
dim1, dim2	characters to indicate the PCs to plot on x-axis (dim1) and y-axis (dim2), e.g. dim1='PC1', dim2='PC2'; dim1='PC2', dim2='PC3'; etc.
groups	a vector of characters to specify the groups of conditions. The scatter points will be coloured according to provided group information.
plot.title	the title to show on the plot.
ellipse.type	add colour shadow to the sample groups. Options are: "none","ellipse" and "polygon".
add.label	logical, whether to add sample labels to the scatter points.
adj.label	logical, whether to adjust the position of sample labels to avoid overlapping.

Value

a plot in ggplot format.

Examples

```
data(exp.data)
library(edgeR)
library(RUVSeq)
library(ggplot2)
trans_counts <- exp.data$trans_counts

##------> sample informationsamples <- exp.data$samples
samples <- exp.data$samples[,c('condition','brep','srep')]

##------> sum sequencing replicates
idx <- paste0(samples$condition,'_',samples$brep)
y <- sumarrays(trans_counts,group = idx)

##------> update sample information after sum
samples <- samples[samples$srep==samples$srep[1],]

##------> normalisation
dge <- DGEList(counts=y)
dge <- calcNormFactors(dge)
data2pca <- t(counts2CPM(obj = dge,Log = T))

##------> plot PCA
groups <- samples$brep
plotPCAind(data2pca = data2pca,dim1 = 'PC1',dim2 = 'PC2',
            groups = groups,ellipse.type='polygon')

##------> remove batch effects
trans_batch <- remove.batch(read.counts = y,
                      condition = samples$condition,
                      method = 'RUVr')

##------> normalisation and plot PCA again
dge <- DGEList(counts=trans_batch$normalizedCounts)
dge <- suppressWarnings(calcNormFactors(dge))
data2pca <- t(counts2CPM(obj = dge,Log = T))
plotPCAind(data2pca = data2pca,dim1 = 'PC1',dim2 = 'PC2',
            groups = groups,ellipse.type='polygon')
```

---

plotUpdown *Bar plot of up- and down-regulation*

Description

Bar plot of up- and down-regulation

Usage

```
plotUpdown(data2plot, contrast, plot.title = NULL)
```

Arguments

data2plot	a matrix/data.frame with three columns: "contrast", "regulation" and "number". See examples for details.

contrast	a vector of contrast groups, e.g. c('C-A','B-A').
plot.title	the the plot titile.

Value

a bar plot in ggplot format.

Examples

```
data2plot <- data.frame(contrast = c('T10-T2','T10-T2','T19-T2','T19-T2'),
                   regulation = c('down_regulate','up_regulate','down_regulate','up_regulate'),
                        number = c(305,727,1062,1805)
                        )
plotUpdown(data2plot = data2plot,contrast = c('T10-T2','T19-T2'),plot.title = 'Up-down regulation')
```

---

remove.batch *Estimate batch effects between biological replicates*

Description

Estimate batch effects between biological replicates

Usage

```
remove.batch(read.counts, condition, design = NULL, contrast = NULL,
  group = NULL, method = c("RUVr", "RUVg", "RUVs"), cIdx = NULL,
  k = 1, ...)
```

Arguments

read.counts	data.frame/matrix of read counts.
condition	condition labels of the columns in read count data, e.g. c('A','A','A','B','B','B','C','C','C').
design	design matrix relating to conditions to be preserved. If NULL, design matrix will beg generated from provided condition.
contrast	a vector of contrast groups for condition comparisons, e.g. c('B-A','C-A').
group	a vector or factor giving the experimental group/condition for each sample/library. See DGEList for details.
method	one of "RUVr", "RUVg" and "RUVs". See the RUVSeq R package for details http://bioconductor.org/packages/release/bioc/html/RUVSeq.html.
cIdx	a vector of character, logical or numeric to indicate the subset of genes to be used as negative controls in the estimation of the factors of unwanted variation. Default is NULL.
k	a integer number of factors of unwanted variation to be estimated from the data.
...	additional arguments passed to DGEList.

Details

The provided arguments `condition`, `design`, `contrast` and `group` are used to fit a GLM model in `edgeR`, which generates residuals for the `RUVr` method. If the negative controls (not differential expressed genes/transcripts) `cIdx=NULL`, the targets with p-value > 0.1 from `glmQLFit` will be used as negative controls. See the packages `edgeR` and `RUVSeq` for detials.

Value

A list object with elements:

- `W`: the biological-replicates-by-factors matrix with the estiamted factors of batch effects, which can be passed to the design matrix of linear regression as batch effect terms.
- `normalizedCounts`: a expression matrix the same dimension as the input read counts, in which the batch effects have been removed according to the estimated factors.
- `method`: the method used to estimate the batch effects (RUVr, RUVg or RUVs).

Examples

```
samples <- exp.data$samples

##------> sum sequencing replicates
trans_counts <- exp.data$trans_counts
idx <- paste0(samples$condition,'_',samples$brep)
y <- sumarrays(trans_counts,group = idx)

##------> update sample information after sum
samples <- samples[samples$srep==samples$srep[1],]

##------> normalisation
dge <- DGEList(counts=y)
dge <- calcNormFactors(dge)
data2pca <- t(counts2CPM(obj = dge,Log = T))

##------> plot PCA
groups <- samples$brep
plot.PCA.ind(data2pca = data2pca,dim1 = 'PC1',dim2 = 'PC2',
             groups = groups,ellipse.type='polygon')

##------> remove batch effects
y.new <- remove.batch(read.counts = y,
                      condition = samples$condition,
                      method = 'RUVr')

##------> normalisation and plot PCA again
dge <- DGEList(counts=genes_batch$normalizedCounts)
dge <- suppressWarnings(calcNormFactors(dge))
data2pca <- t(counts2CPM(obj = dge,Log = T))
plot.PCA.ind(data2pca = data2pca,dim1 = 'PC1',dim2 = 'PC2',
             groups = groups,ellipse.type='polygon')
```

rowmean — *Calculate column mean of a matrix or data frame based on a grouping variable*

Description

Compute column means across rows of a numeric matrix-like object for each level of a grouping variable.

Usage

```
rowmean(x, group, reorder = F, na.rm = T)
```

Arguments

x	a matrix or data frame.
group	a vector of factor giving grouping, with one element per row of x.
reorder	if TRUE, then the result will be in order of `sort(unique(group))`.
na.rm	logical (TRUE or FALSE). Should NA (including NaN) values be replaced by value 0?

Value

rowmean returns a matrix or data frame containing the means. There will be one row per unique value of group.

See Also

rowsum, rowratio

Examples

```
x <- matrix(runif(50), ncol = 5)
group <- sample(1:4, 10, TRUE)
xmean <- rowmean(x, group)
```

rowratio — *Calculate column raito of a matrix or data frame based on a grouping variable*

Description

Compute column ratio across rows of a numeric matrix-like object for each level of a grouping variable.

Usage

```
rowratio(x, group, reorder = F, na.rm = T)
```

Arguments

x	a matrix or data frame.
group	a vector of factor giving grouping, with one element per row of x.
reorder	if TRUE, then the result will be in order of row names of x. If row names of x is null, the results is not reordered.
na.rm	logical (TRUE or FALSE). Should NA (including NaN) values be replaced by value 0?

Value

rowratio returns a matrix or data frame containing the ratios. There will be one row per unique value of group.

See Also

rowsum, rowmean

Examples

```
x <- matrix(runif(50), ncol = 5)
group <- sample(1:4, 10, TRUE)
xratio <- rowratio(x, group)
```

---

set2 *Generate all possible logical relations between set x and y.*

Description

Generate all possible logical relations between set x and y.

Usage

```
set2(x, y)
```

Arguments

x, y	vectors of set x and set y.

Value

a list of three elements: "x.only", "xy" and "y.only".

See Also

plotEulerDiagram and eulerr::euler

Examples

```
x <- letters[1:10]
y <- letters[5:15]
z <- set2(x,y)
combo <- c(x=length(z$x.only),y=length(z$y.only),"x&y"=length(z$xy))
plotEulerDiagram(combo)
```

`sumarrays` *Sum Over Replicate Arrays*

Description

Sum Over Replicate Arrays

Usage

```
sumarrays(x, group = NULL)
```

Arguments

`x`	a matrix-like object.
`group`	grouping of sample identifier.

Value

A data object of the same class as x with a column for sums according grouping.

Examples

```
set.seed(100)
x<- matrix(rnorm(8*4),8,4)
colnames(x) <- c("a","a","b","b")
sumarrays(x)
data.frame(a=x[,1]+x[,2],b=x[,3]+x[,4])
```

`summaryDAStarget` *Summary DAS genes and DTU transcripts*

Description

Summary DAS genes and DTU transcripts

Usage

```
summaryDAStarget(stat, lfc, cutoff = c(adj.pval = 0.01, deltaPS = 0.1))
```

Arguments

stat	a data.frame object with first column of "target", second column of "contrast", third and fourth columns of DAS gene/DTU transcript statistics (i.e. "adj.pval" and "maxdeltaPS"/"deltaPS", respectively).
cutoff	a numeric vector of cut-offs for the statistics.

Value

a data.frame object, which is a subset of input `stat` after applying the `cutoff`.

---

summaryDEtarget *Summary DE genes and transcripts*

Description

Summary DE genes and transcripts

Usage

```
summaryDEtarget(stat, cutoff = c(adj.pval = 0.01, log2FC = 1))
```

Arguments

stat	a data.frame object with first column of "target", second column of "contrast", third and fourth columns of DE gene/transcript statistics (i.e. "adj.pval" and "log2FC", respectively).
cutoff	a numeric vector of cut-offs to "adj.pval" and "log2FC".

Value

a data.frame object, which is a subset of input `stat` after applying the `cutoff`.

---

summaryStat *Merge two testing statistics*

Description

Merge two testing statistics

Usage

```
summaryStat(x, y, target, contrast = NULL, stat.type = c("adj.pval",
  "lfc"), srot.by = stat.type[1])
```

Arguments

x, y	data.frame object of statistics (e.g. p-values, log2-fold changes, deltaPS), with rows of targets and columns statistics in contrast groups.
contrast	a vector of contrast groups, e.g. `contrast = c('B-A','C-A')`. If NULL, the column names of x are used as contrast.
stat.type	a vector of statistic types of x and y, respectively, e.g. `stat.type = c('adj.pval','lfc')`. stat.type is passed to third and fourth column names of the output data.frame.
srot.by	a column name to sort the output data.frame.

Value

summaryStat returns a data.frame object with first column of target, second column of contrast groups, third column name of statistics x and fourth column of statistic y.

ThreeDRNAseq.app *The shiny app for 3D analysis*

Description

The shiny app for 3D analysis

Usage

```
ThreeDRNAseq.app(data.size.max = 300)
```

Arguments

data.size.max	maximum size limit for unload files in Shiny. Default is 300 (MB).

Value

the Shiny App.

See Also

shiny

Examples

```
RNAseq3D.app()
```

TPM.filter *Filter low expression based on TPM (transcript per million reads)*

Description

Filter low expression based on TPM (transcript per million reads)

Usage

```
TPM.filter(TPM, sample.n = 3, tpm.cut = 1)
```

Arguments

TPM	a data.frame/matrix of TPM.
sample.n	number of samples
tpm.cut	a numeric value of TPM cut-off

Details

An expressed target must have $\geq$ sample.n with TPM $\geq$ tpm.cut.

Value

A vector of logcial values (TRUE/FALSE) to indicate the rows of the input data to keep/filter.

transAbundance2PS *Calculate transcript PS (percent spliced) and ΔPS from abundance*

Description

Calculate transcript PS (percent spliced) and ΔPS from abundance

Usage

```
transAbundance2PS(transAbundance = NULL, PS = NULL, contrast,
  condition, mapping)
```

Arguments

transAbundance	matrix of transcript level abundance, e.g. TPM and read count exprssion.
PS	PS value matrix. If a PS matrix provided, the ΔPS values is directly calculated from the PS matrix based on the contrast groups, otherwise, new PS value will be generate. Default is NULL.
contrast	a vector of contrast groups to calculate ΔPS values, e.g. c('B-A','C-A'), which compares conditions "B" and "C" to condition "A".
condition	a vector of condition labels which match the columns in the abundance matrix, e.g. c('A','A','A','B','B','B','C','C','C'), which means the abundance matrix corresponds to 3 conditions and each condition has 3 replicates.
mapping	transcript-gene mapping matrix. First column is transcript list with column name "TXNAME" and second column is gene list with column name "GENEID".

Details

PS is defined as the ratio of transcript abundance divided by the gene abundance (sum of all transcript abundance of the same gene).

Value

a list of two data.frames: "PS" and "deltaPS".

Examples

```
set.seed(36)
transAbundance <- matrix(rnorm(36),ncol = 9,nrow = 3)
PS <- NULL
contrast <- c('B-A','C-A')
condition <- c('A','A','A','B','B','B','C','C','C')
mapping <- data.frame(TXNAME=c('trans1.1','trans1.2','trans2'),
                      GENEID=c('gene1','gene1','gene2'))
transAbundance2PS(transAbundance = transAbundance,PS = PS,
                  contrast = contrast,condition = condition,mapping = mapping)
```

---

trend.mean.variance *Estimate mean-variance trend of expression*

Description

Estimate mean-variance trend of expression

Usage

```
trend.mean.variance(obj, design, lib.size = NULL, span = 0.5, ...)
```

Arguments

obj	a matrix of gene/transcript read counts, or a DGEList object. Read counts must be non-negative and NAs are not permitted.
design	design matrix with rows of samples and columns of conditions to fit a linear model.
lib.size	a vector of library size for each sample. If NULL, the library size will be either extracted from the DGEList object or calculated by summing up all the reads of genes/transcripts in each sample.
span	a numeric value passed to lowess smoothing window as a proportion to fit the mean-variance trend.
...	additional arguments passed to lmFit function.

Value

a list object with elements:

- fit: the linear regression results of expression based on design matrix.
- sx: a numeric vector of mean values.
- sy: a numeric vector of variance values.
- l: the lowess fit result based on sx and sy.

See Also

voom, lowess and condition2design.

Index