Package ‘variancePartition’

March 23, 2017

Type Package

Title Quantify and interpret divers of variation in multilevel gene expression experiments

Version 1.4.2

Date 2016-11-29

Author Gabriel E. Hoffman

Maintainer Gabriel E. Hoffman <gabriel.hoffman@mssm.edu>

Description Quantify and interpret multiple sources of biological and technical variation in gene expression experiments. Uses linear mixed model to quantify variation in gene expression attributable to individual, tissue, time point, or technical variables.

VignetteBuilder knitr

License GPL (>= 2)

Suggests edgeR, dendextend, tximport, tximportData, ballgown, DESeq2, readr, knitr, BiocStyle

biocViews RNASeq, GeneExpression, Regression, Software

Depends ggplot2, foreach, Biobase, methods

Imports MASS, pbkrtest (>= 0.4-4), iterators, splines, colorRamps, gplots, reshape2, lme4 (>= 1.1-10), doParallel, limma, grDevices, graphics, utils, stats

RoxygenNote 6.0.1

NeedsCompilation no

R topics documented:

as.matrix, varPartResults-method .............................................. 2
calcVarPart ................................................................. 3
canCorPairs ................................................................. 4
colinearityScore ............................................................. 5
ESS .............................................................................. 6
extractVarPart ............................................................ 7
fitExtractVarPartModel ....................................................... 8
fitVarPartModel ............................................................ 11
getVarianceComponents .................................................... 13
ggColorHue ................................................................. 14
as.matrix, varPartResults-method

Convert to matrix

Description
Convert varPartResults to matrix

Usage

```r
## S4 method for signature 'varPartResults'
as.matrix(x, ...)
```

Arguments

- `x` varPartResults
- `...` other arguments.

Value

matrix

Examples

```r
# load library
# library(variancePartition)

# load simulated data:
# geneExpr: matrix of gene expression values
# info: information/metadata about each sample
data(varPartData)

# Specify variables to consider
# Age is continuous so we model it as a fixed effect
# Individual and Tissue are both categorical, so we model them as random effects
form <- ~ Age + (1|Individual) + (1|Tissue)

# Fit model
varPart <- fitExtractVarPartModel( geneExpr[1:5,], form, info )

# convert to matrix
```
**Description**

Compute fraction of variation attributable to each variable in regression model. Also interpretable as the intra-class correlation after correcting for all other variables in the model.

**Usage**

```r
calcVarPart(fit, adjust = NULL, adjustAll = FALSE, showWarnings = TRUE, ...)
```

## S4 method for signature 'lm'
```r
calcVarPart(fit, adjust = NULL, adjustAll = FALSE, showWarnings = TRUE, ...)
```

## S4 method for signature 'lmerMod'
```r
calcVarPart(fit, adjust = NULL, adjustAll = FALSE, showWarnings = TRUE, ...)
```

**Arguments**

- `fit` : model fit from lm() or lmer()
- `adjust` : remove variation from specified variables from the denominator. This computes the adjusted ICC with respect to the specified variables
- `adjustAll` : adjust for all variables. This computes the adjusted ICC with respect to all variables
- `showWarnings` : show warnings about model fit (default TRUE)
- `...` : additional arguments (not currently used)

**Value**

fraction of variance explained / ICC for each variable in the model

**Examples**

```r
library(lme4)
data(varPartData)

# Linear mixed model
fit <- lmer( geneExpr[1,] ~ (1|Tissue) + Age, info)
calcVarPart( fit )

# Linear model
# Note that the two models produce slightly different results
# This is expected; they are different statistical estimates
# of the same underlying value
canCorPairs

```r
fit <- lm( geneExpr[,1] ~ Tissue + Age, info)
calcVarPart( fit )
```

description

Assess correlation between all pairs of variables in a formula

Usage

canCorPairs(formula, data)

Arguments

- **formula**: standard linear model formula (doesn’t support random effects currently, so just change the syntax)
- **data**: data.frame with the data for the variables in the formula

Details

Canonical Correlation Analysis (CCA) is similar to correlation between two vectors, except that CCA can accommodate matrices as well. For a pair of variables, canCorPairs assesses the degree to which they co-vary and contain the same information. Variables in the formula can be a continuous variable or a discrete variable expanded to a matrix (which is done in the backend of a regression model). For a pair of variables, canCorPairs uses CCA to compute the correlation between these variables and returns the pairwise correlation matrix.

Statistically, let rho be the array of correlation values returned by the standard R function cancor to compute CCA. canCorPairs returns rho / sum(rho) which is the fraction of the maximum possible correlation.

Note that CCA returns correlations values between 0 and 1

Value

Matrix of correlation values between all pairs of variables.

Examples

```r
# load library
# library(variancePartition)

# load simulated data:
data(varPartData)

# specify formula
form <- ~ Individual + Tissue + Batch + Age + Height

# Compute Canonical Correlation Analysis (CCA)
# between all pairs of variables
colinearityScore

# returns absolute correlation value
C = canCorPairs( form, info)

# Plot correlation matrix
plotCorrMatrix( C )

colinearityScore  
Collinearity score

Description
Collinearity score for a regression model indicating if variables are too highly correlated to give meaningful results

Usage
colinearityScore(fit)

Arguments
fit  
regression model fit from lm() or lmer()

Value
Returns the collinearity score between 0 and 1, where a score > 0.999 means the degree of collinear-
ity is too high. This function reports the correlation matrix between coefficient estimates for fixed
effects. The collinearity score is the maximum absolute correlation value of this matrix. Note
that the values are the correlation between the parameter estimates, and not between the variables
themselves.

Examples

# load library
# library(variancePartition)

# load simulated data: 
data(varPartData)
form <- ~ Age + (1|Individual) + (1|Tissue)
res <- fitVarPartModel( geneExpr[1:10,], form, info )

# evaluate the collinearity score on the first model fit
# this reports the correlation matrix between coefficients estimates
# for fixed effects
# the collinearity score is the maximum absolute correlation value
# If the collinearity score > .999 then the variance partition
# estimates may be problematic
# In that case, a least one variable should be omitted
colinearityScore(res[[1]])
## ESS

### Effective sample size

#### Description

Compute effective sample size based on correlation structure in linear mixed model

#### Usage

```r
ESS(fit, method = "full")
```

```r
## S4 method for signature 'lmerMod'
ESS(fit, method = "full")
```

#### Arguments

- **fit**: model fit from `lmer()`
- **method**: "full" uses the full correlation structure of the model. The "approximate" method makes the simplifying assumption that the study has a mean of m samples in each of k groups, and computes m based on the study design. When the study design is evenly balanced (i.e., the assumption is met), this gives the same results as the "full" method.

#### Details


- **"full" method**: if \( V_x = \text{var}(Y;x) \) is the variance-covariance matrix of Y, the response, based on the covariate x, then the effective sample size corresponding to this covariate is \( \Sigma_{i,j} (V_x^{-1})_{i,j} \). In R notation, this is: \( \text{sum}(\text{solve}(V_x)) \). In practice, this can be evaluated as \( \text{sum}(w) \), where R

- **"approximate" method**: Letting m be the mean number of samples per group, k be the number of groups, and \( \rho \) be the intraclass correlation, the effective sample size is \( m^k / (1+\rho*(m-1)) \)

Note that these values are equal when there are exactly m samples in each group. If m is only an average then this is an approximation.

#### Value

effective sample size for each random effect in the model

#### Examples

```r
library(lme4)
data(varPartData)

# Linear mixed model
fit <- lmer(geneExpr[,1] ~ (1|Individual) + (1|Tissue) + Age, info)

# Effective sample size
ESS(fit)
```
**Description**

Extract variance statistics from list of models fit with `lm()` or `lmer()`

**Usage**

```r
extractVarPart(modelList, adjust = NULL, adjustAll = FALSE, showWarnings = TRUE, ...)
```

**Arguments**

- `modelList`: list of `lmer()` model fits
- `adjust`: remove variation from specified variables from the denominator. This computes the adjusted ICC with respect to the specified variables
- `adjustAll`: adjust for all variables. This computes the adjusted ICC with respect to all variables. This overrides the previous argument, so all variables are included in adjustment.
- `showWarnings`: show warnings about model fit (default `TRUE`)
- `...`: other arguments

**Value**

data.frame of fraction of variance explained by each variable, after correcting for all others.

**Examples**

```r
# library(variancePartition)

# optional step to run analysis in parallel on multicore machines
# Here, we used 4 threads
library(doParallel)
c1 <- makeCluster(4)
registerDoParallel(c1)
# or by using the doSNOW package

# load simulated data:
# geneExpr: matrix of gene expression values
# info: information/metadata about each sample
data(varPartData)

# Specify variables to consider
# Age is continuous so we model it as a fixed effect
# Individual and Tissue are both categorical, so we model them as random effects
form <- ~ Age + (1|Individual) + (1|Tissue)

# Step 1: fit linear mixed model on gene expression
# If categorical variables are specified, a linear mixed model is used
# If all variables are modeled as continuous, a linear model is used
# each entry in results is a regression model fit on a single gene
```
# Step 2: extract variance fractions from each model fit
# for each gene, returns fraction of variation attributable to each variable
# Interpretation: the variance explained by each variable
# after correction for all other variables
varPart <- fitExtractVarPartModel( geneExpr, form, info )

# violin plot of contribution of each variable to total variance
plotVarPart( sortCols( varPart ) )

# Advanced:
# Fit model and extract variance in two separate steps
# Step 1: fit model for each gene, store model fit for each gene in a list
results <- fitVarPartModel( geneExpr, form, info )

# Step 2: extract variance fractions
varPart <- extractVarPart( results )

# stop cluster
stopCluster(cl)

---

**fitExtractVarPartModel**

*Fit linear (mixed) model, report variance fractions*

**Description**

Fit linear (mixed) model to estimate contribution of multiple sources of variation while simultaneously correcting for all other variables. Report fraction of variance attributable to each variable.

**Usage**

```r
fitExtractVarPartModel(exprObj, formula, data, REML = FALSE,
useWeights = TRUE, weightsMatrix = NULL, adjust = NULL,
adjustAll = FALSE, showWarnings = TRUE,
control = lme4::lmerControl(calc.derivs = FALSE, check.rankX =
"stop.deficient"), ...)
```

## S4 method for signature 'matrix'

```r
fitExtractVarPartModel(exprObj, formula, data, REML = FALSE, useWeights = TRUE, weightsMatrix = NULL, adjust = NULL,
adjustAll = FALSE, showWarnings = TRUE,
control = lme4::lmerControl(calc.derivs = FALSE, check.rankX =
"stop.deficient"), ...)
```

## S4 method for signature 'data.frame'

```r
fitExtractVarPartModel(exprObj, formula, data, REML = FALSE, useWeights = TRUE, weightsMatrix = NULL, adjust = NULL,
adjustAll = FALSE, showWarnings = TRUE,
control = lme4::lmerControl(calc.derivs = FALSE, check.rankX =
"stop.deficient"), ...)
```
fitExtractVarPartModel

## S4 method for signature 'EList'
fitExtractVarPartModel(exprObj, formula, data, REML = FALSE,
useWeights = TRUE, weightsMatrix = NULL, adjust = NULL,
adjustAll = FALSE, showWarnings = TRUE,
control = lme4::lmerControl(calc.derivs = FALSE, check.rankX =
"stop.deficient"), ...)

## S4 method for signature 'ExpressionSet'
fitExtractVarPartModel(exprObj, formula, data,
REML = FALSE, useWeights = TRUE, weightsMatrix = NULL, adjust = NULL,
adjustAll = FALSE, showWarnings = TRUE,
control = lme4::lmerControl(calc.derivs = FALSE, check.rankX =
"stop.deficient"), ...)

Arguments

exprObj matrix of expression data (g genes x n samples), or ExpressionSet, or EList returned by voom() from the limma package

formula specifies variables for the linear (mixed) model. Must only specify covariates, since the rows of exprObj are automatically used a a response. e.g.: ∼ a + b + (1|c)

data data.frame with columns corresponding to formula

REML use restricted maximum likelihood to fit linear mixed model. default is FALSE. Strongly discourage against changing this option

useWeights if TRUE, analysis uses heteroskedastic error estimates from voom(). Value is ignored unless exprObj is an EList() from voom() or weightsMatrix is specified

weightsMatrix matrix the same dimension as exprObj with observation-level weights from voom(). Used only if useWeights is TRUE

adjust remove variation from specified variables from the denominator. This computes the adjusted ICC with respect to the specified variables

adjustAll adjust for all variables. This computes the adjusted ICC with respect to all variables. This overrides the previous argument, so all variables are include in adjust.

showWarnings show warnings about model fit (default TRUE)

control control settings for lmer()

... Additional arguments for lmer() or lm()

Details

A linear (mixed) model is fit for each gene in exprObj, using formula to specify variables in the regression. If categorical variables are modeled as random effects (as is recommended), then a linear mixed model us used. For example if formula is ∼ a + b + (1|c), then to model is

fit <- lmer( exprObj[,j] ∼ a + b + (1|c), data=data)

If there are no random effects, so formula is ∼ a + b + c, a 'standard' linear model is used:

fit <- lm( exprObj[,j] ∼ a + b + c, data=data)

In both cases, useWeights=TRUE causes weightsMatrix[j,] to be included as weights in the regression model.
Note: Fitting the model for 20,000 genes can be computationally intensive. To accelerate computation, models can be fit in parallel using foreach/dopar to run loops in parallel. Parallel processing must be enabled before calling this function. See below.

The regression model is fit for each gene separately. Samples with missing values in either gene expression or metadata are omitted by the underlying call to lm/lmer.

Value

list() of where each entry is a model fit produced by lmer() or lm()

Examples

```r
# load library
library(variancePartition)

# optional step to run analysis in parallel on multicore machines
# Here, we used 4 threads
library(doParallel)
cl <- makeCluster(4)
registerDoParallel(cl)

# or by using the doSNOW package

# load simulated data:
# geneExpr: matrix of gene expression values
# info: information/metadata about each sample
data(varPartData)

# Specify variables to consider
# Age is continuous so we model it as a fixed effect
# Individual and Tissue are both categorical, so we model them as random effects
form <- ~ Age + (1|Individual) + (1|Tissue)

# Step 1: fit linear mixed model on gene expression
# If categorical variables are specified, a linear mixed model is used
# If all variables are modeled as continuous, a linear model is used
# each entry in results is a regression model fit on a single gene
# Step 2: extract variance fractions from each model fit
# for each gene, returns fraction of variation attributable to each variable
# Interpretation: the variance explained by each variable after correction for all other variables
varPart <- fitExtractVarPartModel( geneExpr, form, info )

# violin plot of contribution of each variable to total variance
plotVarPart( sortCols( varPart ) )

# Note: fitExtractVarPartModel also accepts ExpressionSet
data(sample.ExpressionSet, package="Biobase")

# ExpressionSet example
form <- ~ (1|sex) + (1|type) + score
info2 <- pData(sample.ExpressionSet)
varPart2 <- fitExtractVarPartModel( sample.ExpressionSet, form, info2 )

# stop cluster
stopCluster(cl)
```
fitVarPartModel

Description

Fit linear (mixed) model to estimate contribution of multiple sources of variation while simultaneously correcting for all other variables.

Usage

fitVarPartModel(exprObj, formula, data, REML = FALSE, useWeights = TRUE, weightsMatrix = NULL, showWarnings = TRUE, fxn = identity, control = lme4::lmerControl(calc.derivs = FALSE, check.rankX = "stop.deficient"), ...)

## S4 method for signature 'matrix'
fittedVarPartModel(exprObj, formula, data, REML = FALSE, useWeights = TRUE, weightsMatrix = NULL, showWarnings = TRUE, fxn = identity, control = lme4::lmerControl(calc.derivs = FALSE, check.rankX = "stop.deficient"), ...)

## S4 method for signature 'data.frame'
fittedVarPartModel(exprObj, formula, data, REML = FALSE, useWeights = TRUE, weightsMatrix = NULL, showWarnings = TRUE, fxn = identity, control = lme4::lmerControl(calc.derivs = FALSE, check.rankX = "stop.deficient"), ...)

## S4 method for signature 'EList'
fittedVarPartModel(exprObj, formula, data, REML = FALSE, useWeights = TRUE, weightsMatrix = NULL, showWarnings = TRUE, fxn = identity, control = lme4::lmerControl(calc.derivs = FALSE, check.rankX = "stop.deficient"), ...)

## S4 method for signature 'ExpressionSet'
fittedVarPartModel(exprObj, formula, data, REML = FALSE, useWeights = TRUE, weightsMatrix = NULL, showWarnings = TRUE, fxn = identity, control = lme4::lmerControl(calc.derivs = FALSE, check.rankX = "stop.deficient"), ...)

Arguments

exprObj matrix of expression data (g genes x n samples), or ExpressionSet, or EList returned by voom() from the limma package

formula specifies variables for the linear (mixed) model. Must only specify covariates, since the rows of exprObj are automatically used as a response. e.g.: ~ a + b + (1|c)

data data.frame with columns corresponding to formula
REML  use restricted maximum likelihood to fit linear mixed model. default is FALSE. Strongly discourage against changing this option
useWeights  if TRUE, analysis uses heteroskedastic error estimates from voom(). Value is ignored unless exprObj is an EList() from voom() or weightsMatrix is specified
weightsMatrix  matrix the same dimension as exprObj with observation-level weights from voom(). Used only if useWeights is TRUE
showWarnings  show warnings about model fit (default TRUE)
fxn  apply function to model fit for each gene. Defaults to identify function so it returns the model fit itself
control  control settings for lmer()

...  Additional arguments for lmer() or lm()

Details

A linear (mixed) model is fit for each gene in exprObj, using formula to specify variables in the regression. If categorical variables are modeled as random effects (as is recommended), then a linear mixed model us used. For example if formula is ~ a + b + (1|c), then to model is

fit <- lmer(exprObj[,] ~ a + b + (1|c), data=data)

If there are no random effects, so formula is ~ a + b + c, a 'standard' linear model is used:

fit <- lm(exprObj[,] ~ a + b + c, data=data)

In both cases, useWeights=TRUE causes weightsMatrix[j,j] to be included as weights in the regression model.

Note: Fitting the model for 20,000 genes can be computationally intensive. To accelerate computation, models can be fit in parallel using foreach/dopar to run loops in parallel. Parallel processing must be enabled before calling this function. See below.

The regression model is fit for each gene separately. Samples with missing values in either gene expression or metadata are omitted by the underlying call to lm/lmer.

Since this function returns a list of each model fit, using this function is slower and uses more memory than fitExtractVarPartModel().

Value

list() of where each entry is a model fit produced by lmer() or lm()

Examples

# load library
# library(variancePartition)

# optional step to run analysis in parallel on multicore machines
# Here, we used 4 threads
library(doParallel)
c1 <- makeCluster(4)
registerDoParallel(c1)
# or by using the doSNOW package

# load simulated data:
# geneExpr: matrix of gene expression values
# info: information/metadata about each sample
getVarianceComponents

Extract variance terms

Description

Extract variance terms from a model fit with lm() or lmer()

Usage

getVarianceComponents(fit)

Arguments

fit list of lmer() model fits
Value

variance explained by each variable

Examples

# library(variancePartition)

# optional step to run analysis in parallel on multicore machines
# Here, we used 4 threads
library(doParallel)
cl <- makeCluster(4)
registerDoParallel(cl)
# or by using the doSNOW package

# load simulated data:
# geneExpr: matrix of gene expression values
# info: information/metadata about each sample
data(varPartData)

# Specify variables to consider
# Age is continuous so we model it as a fixed effect
# Individual and Tissue are both categorical, so we model them as random effects
form <- ~ Age + (1|Individual) + (1|Tissue)

# Fit model and extract variance in two separate steps
# Step 1: fit model for each gene, store model fit for each gene in a list
modellist <- fitVarPartModel( geneExpr, form, info )
fit <- modellist[[1]]
getVarianceComponents( fit )

# stop cluster
stopCluster(cl)

---

**ggColorHue**

*Default colors for ggplot*

Description

Return an array of n colors the same as the default used by ggplot2

Usage

`ggColorHue(n)`

Arguments

n number of colors

Value

array of colors of length n
Description

Plot correlation matrix

Usage

plotCorrMatrix(C, dendrogram = "both", sort = TRUE, margins = c(13, 13),
key.xlab = "correlation", ...)

Arguments

C correlation matrix: R or R^2 matrix
dendrogram character string indicating whether to draw 'both' or none'
sort sort rows and columns based on clustering
margins spacing of plot
key.xlab label of color gradient
... additional arguments to heatmap.2

Details

Plots image of correlation matrix using customized call to heatmap.2

Value

Image of correlation matrix

Examples

# simulate simple matrix of 10 variables
mat = matrix(rnorm(1000), ncol=10)

# compute correlation matrix
C = cor(mat)

# plot correlations
plotCorrMatrix( C )

# plot squared correlations
plotCorrMatrix( C^2, dendrogram="none" )
plotCorrStructure

Description
Plot correlation structure of a gene based on random effects

Usage
plotCorrStructure(fit, varNames = names(coef(fit)), reorder = TRUE, 
apal = colorRampPalette(c("white", "red", "darkred")),
hclust.method = "complete")

Arguments
fit          linear mixed model fit of a gene produced by lmer() or fitVarPartModel()
varNames     variables in the metadata for which the correlation structure should be shown. Variables must be random effects
reorder      how to reorder the rows/columns of the correlation matrix. reorder=FALSE gives no reorder. reorder=TRUE reorders based on hclust. reorder can also be an array of indices to reorder the samples manually
pal          color palette
hclust.method clustering methods for hclust

Value
Image of correlation structure between each pair of experiments for a single gene

Examples

# load library
# library(variancePartition)

# optional step to run analysis in parallel on multicore machines
# Here, we used 4 threads
library(doParallel)
c1 <- makeCluster(4)
registerDoParallel(c1)
# or by using the doSNOW package

# load simulated data:
data(varPartData)

# specify formula
form <- ~ Age + (1|Individual) + (1|Tissue)

# fit and return linear mixed models for each gene
fitList <- fitVarPartModel( geneExpr[1:10,], form, info )

# Focus on the first gene
fit = fitList[[1]]
# plot correlation structure based on Individual, reordering samples with hclust
plotCorrStructure( fit, "Individual" )

# don't reorder
plotCorrStructure( fit, "Individual", reorder=FALSE )

# plot correlation structure based on Tissue, reordering samples with hclust
plotCorrStructure( fit, "Tissue" )

# don't reorder
plotCorrStructure( fit, "Tissue", FALSE )

# plot correlation structure based on all random effects
# reorder manually by Tissue and Individual
idx = order(info$Tissue, info$Individual)
plotCorrStructure( fit, reorder=idx )

# plot correlation structure based on all random effects
# reorder manually by Individual, then Tissue
idx = order(info$Individual, info$Tissue)
plotCorrStructure( fit, reorder=idx )

# stop cluster
stopCluster(cl)

---

**plotPercentBars**  
*Bar plot of variance fractions*

**Description**  
Bar plot of variance fractions for a subset of genes

**Usage**  
```
plotPercentBars(varPart, col = c(ggColorHue(ncol(varPart) - 1), "grey85"))
```

**Arguments**  
- `varPart`: object returned by `extractVarPart()` or `fitExtractVarPartModel()`  
- `col`: color of bars for each variable

**Value**  
Returns ggplot2 barplot

**Examples**  
```
# library(variancePartition)

# optional step to run analysis in parallel on multicore machines
# Here, we used 4 threads
library(doParallel)
```
cl <- makeCluster(4)
registerDoParallel(cl)
# or by using the doSNOW package

# load simulated data:
# geneExpr: matrix of gene expression values
# info: information/metadata about each sample
data(varPartData)

# Specify variables to consider
form <- ~ Age + (1|Individual) + (1|Tissue)

# Fit model
varPart <- fitExtractVarPartModel( geneExpr, form, info )

# Bar plot for a subset of genes showing variance fractions
plotPercentBars( varPart[1:5,] )

# Move the legend to the top
plotPercentBars( varPart[1:5,] ) + theme(legend.position="top")

# stop cluster
stopCluster(cl)


plotStratify

plotStratify

**Description**

Plot gene expression stratified by another variable

**Usage**

```
plotStratify(formula, data, xlab, ylab, main, sortBy, colorBy, sort = TRUE, 
text = NULL, text.y = 1, text.size = 5, pts.cex = 1, ylim = NULL, 
legend = TRUE, x.labels = FALSE)
```

**Arguments**

- `formula`: specify variables shown in the x- and y-axes. Y-axis should be continuous variable, x-axis should be discrete.
- `data`: data.frame storing continuous and discrete variables specified in formula
- `xlab`: label x-asis. Defaults to value of xval
- `ylab`: label y-asis. Defaults to value of yval
- `main`: main label
- `sortBy`: name of column in geneExpr to sort samples by. Defaults to xval
- `colorBy`: name of column in geneExpr to color box plots. Defaults to xval
- `sort`: if TRUE, sort boxplots by median value, else use default ordering
- `text`: plot text on the top left of the plot
- `text.y`: indicate position of the text on the y-axis as a fraction of the y-axis range
plotStratifyBy

text.size size of text
pts.cex size of points
ylim specify range of y-axis
legend show legend
x.labels show x axis labels

Value

ggplot2 object

Examples

# Note: This is a newer, more convient interface to plotStratifyBy()

# load library
# library(variancePartition)

# load simulated data:
data(varPartData)

# Create data.frame with expression and Tissue information for each sample
GE = data.frame( Expression = geneExpr[1,], Tissue = info$Tissue)

# Plot expression stratified by Tissue
plotStratify( Expression ~ Tissue, GE )

# Omit legend and color boxes grey
plotStratify( Expression ~ Tissue, GE, colorBy = NULL)

# Specify colors
col = c( B="green", A="red", C="yellow")
plotStratify( Expression ~ Tissue, GE, colorBy=col, sort=FALSE)
plotStratifyBy

**Arguments**

- `geneExpr`: data.frame of gene expression values and another variable for each sample. If there are multiple columns, the user can specify which one to use.
- `xval`: name of column in `geneExpr` to be used along x-axis to stratify gene expression.
- `yval`: name of column in `geneExpr` indicating gene expression.
- `xlab`: label x-asis. Defaults to value of `xval`.
- `ylab`: label y-asis. Defaults to value of `yval`.
- `main`: main label.
- `sortBy`: name of column in `geneExpr` to sort samples by. Defaults to `xval`.
- `colorBy`: name of column in `geneExpr` to color box plots. Defaults to `xval`.
- `sort`: if TRUE, sort boxplots by median value, else use default ordering.
- `text`: plot text on the top left of the plot.
- `text.y`: indicate position of the text on the y-axis as a fraction of the y-axis range.
- `text.size`: size of text.
- `pts.cex`: size of points.
- `ylim`: specify range of y-axis.
- `legend`: show legend.
- `x.labels`: show x axis labels.

**Value**

- ggplot2 object

**Examples**

```r
# load library
# library(variancePartition)

# load simulated data:
data(varPartData)

# Create data.frame with expression and Tissue information for each sample
GE = data.frame( Expression = geneExpr[1,], Tissue = info$Tissue)

# Plot expression stratified by Tissue
plotStratifyBy( GE, "Tissue", "Expression")

# Omit legend and color boxes grey
plotStratifyBy( GE, "Tissue", "Expression", colorBy = NULL)

# Specify colors
col = c( B="green", A="red", C="yellow")
plotStratifyBy( GE, "Tissue", "Expression", colorBy=col, sort=FALSE)
```
plotVarPart

Violin plot of variance fractions

Description

Violin plot of variance fraction for each gene and each variable

Usage

plotVarPart(obj, col = c(ggColorHue(ncol(obj) - 1), "grey85"),
label.angle = 20, main = "", ylab = "", convertToPercent = TRUE, ...)

## S4 method for signature 'matrix'
plotVarPart(obj, col = c(ggColorHue(ncol(obj) - 1),
"grey85"), label.angle = 20, main = "", ylab = "",
convertToPercent = TRUE, ...)

## S4 method for signature 'data.frame'
plotVarPart(obj, col = c(ggColorHue(ncol(obj) - 1),
"grey85"), label.angle = 20, main = "", ylab = "",
convertToPercent = TRUE, ...)

## S4 method for signature 'VarPartResults'
plotVarPart(obj, col = c(ggColorHue(ncol(obj) - 1),
"grey85"), label.angle = 20, main = "", ylab = "",
convertToPercent = TRUE, ...)

Arguments

obj varParFrac object returned by fitExtractVarPart or extractVarPart
col vector of colors
label.angle angle of labels on x-axis
main title of plot
ylab text on y-axis
convertToPercent multiply fractions by 100 to convert to percent values
... additional arguments

Value

Makes violin plots of variance components model. This function uses the graphics interface from ggplot2. Warnings produced by this function usually ggplot2 warning that the window is too small.

Examples

# load library
# library(variancePartition)

# optional step to run analysis in parallel on multicore machines
# Here, we used 4 threads
library(doParallel)
c1 <- makeCluster(4)
registerDoParallel(c1)
# or by using the doSNOW package

# load simulated data:
# geneExpr: matrix of gene expression values
# info: information/metadata about each sample
data(varPartData)

# Specify variables to consider
# Age is continuous so we model it as a fixed effect
# Individual and Tissue are both categorical, so we model them as random effects
form <- ~ Age + (1|Individual) + (1|Tissue)

varPart <- fitExtractVarPartModel(geneExpr, form, info)

# violin plot of contribution of each variable to total variance
plotVarPart(sortCols(varPart))

# stop cluster
stopCluster(cl)

---

residuals,VarParFitList-method

Residuals from model fit

Description

Extract residuals for each gene from model fit with fitVarPartModel()

Usage

```r
## S4 method for signature 'VarParFitList'
residuals(object, ...)
```

Arguments

- `object`: object produced by fitVarPartModel()
- `...`: other arguments.

Details

If model is fit with missing data, residuals returns NA for entries that were missing in the original data

Value

Residuals extracted from model fits stored in object
sortCols

Examples

# load library
# library(variancePartition)

# optional step to run analysis in parallel on multicore machines
# Here, we used 4 threads
library(doParallel)
c1 <- makeCluster(4)
registerDoParallel(c1)
# or by using the doSNOW package

# load simulated data:
# geneExpr: matrix of gene expression values
# info: information/metadata about each sample
data(varPartData)

data(varPartData)

# Specify variables to consider
# Age is continuous so we model it as a fixed effect
# Individual and Tissue are both categorical, so we model them as random effects
form <- ~ Age + (1|Individual) + (1|Tissue)

# Fit model
modelFit <- fitVarPartModel( geneExpr, form, info )

# Extract residuals of model fit
res <- residuals( modelFit )

# stop cluster
stopCluster(cl)

sortCols

Sort variance partition statistics

Description

Sort columns returned by extractVarPart() or fitExtractVarPartModel()

Usage

sortCols(x, FUN = median, decreasing = TRUE, last = c("Residuals", "Measurement.error"), ...)

## S4 method for signature 'matrix'
sortCols(x, FUN = median, decreasing = TRUE, last = c("Residuals", "Measurement.error"), ...)

## S4 method for signature 'data.frame'
sortCols(x, FUN = median, decreasing = TRUE, last = c("Residuals", "Measurement.error"), ...)

## S4 method for signature 'varPartResults'
sortCols(x, FUN = median, decreasing = TRUE, last = c("Residuals", "Measurement.error"), ...)
sortCols

Arguments

- **x**: object returned by extractVarPart() or fitExtractVarPartModel()
- **FUN**: function giving summary statistic to sort by. Defaults to median
- **decreasing**: logical. Should the sorting be increasing or decreasing?
- **last**: columns to be placed on the right, regardless of values in these columns
- **...**: other arguments to sort

Value

data.frame with columns sorted by mean value, with Residuals in last column

Examples

```r
# library(variancePartition)

# optional step to run analysis in parallel on multicore machines
# Here, we used 4 threads
library(doParallel)
cl <- makeCluster(4)
registerDoParallel(cl)
# or by using the doSNOW package

# load simulated data:
# geneExpr: matrix of gene expression values
# info: information/metadata about each sample
data(varPartData)

# Specify variables to consider
# Age is continuous so we model it as a fixed effect
# Individual and Tissue are both categorical, so we model them as random effects
form <- ~ Age + (1|Individual) + (1|Tissue)

# Step 1: fit linear mixed model on gene expression
# If categorical variables are specified, a linear mixed model is used
# If all variables are modeled as continuous, a linear model is used
# each entry in results is a regression model fit on a single gene
# Step 2: extract variance fractions from each model fit
# for each gene, returns fraction of variation attributable to each variable
# Interpretation: the variance explained by each variable
# after correction for all other variables
varPart <- fitExtractVarPartModel( geneExpr, form, info )

# violin plot of contribution of each variable to total variance
# sort columns by median value
plotVarPart( sortCols( varPart ) )

# stop cluster
stopCluster(cl)
```
varPartConfInf

Linear mixed model confidence intervals

Description

Fit linear mixed model to estimate contribution of multiple sources of variation while simultaneously correcting for all other variables. Then perform parametric bootstrap sampling to get a 95% confidence intervals for each variable for each gene.

Usage

\[\text{varPartConfInf}(\text{exprObj}, \text{formula}, \text{data}, \text{REML} = \text{FALSE}, \text{useWeights} = \text{TRUE}, \]
\[\text{weightsMatrix} = \text{NULL}, \text{adjust} = \text{NULL}, \text{adjustAll} = \text{FALSE}, \]
\[\text{showWarnings} = \text{TRUE}, \text{colinearityCutoff} = 0.999, \]
\[\text{control} = \text{lme4>::lmerControl}(\text{calc.derivs} = \text{FALSE}, \text{check.rankX} = \]
\["\text{stop.deficient"}, \text{nsim} = 1000, \ldots)\]

Arguments

- **exprObj**: matrix of expression data (g genes x n samples), or ExpressionSet, or EList returned by voom() from the limma package
- **formula**: specifies variables for the linear (mixed) model. Must only specify covariates, since the rows of exprObj are automatically used a a response. e.g.: \(\sim a + b + (1|c)\)
- **data**: data.frame with columns corresponding to formula
- **REML**: use restricted maximum likelihood to fit linear mixed model. default is FALSE. Strongly discourage against changing this option
- **useWeights**: if TRUE, analysis uses heteroskedastic error estimates from voom(). Value is ignored unless exprObj is an EList() from voom() or weightsMatrix is specified
- **weightsMatrix**: matrix the same dimension as exprObj with observation-level weights from voom(). Used only if useWeights is TRUE
- **adjust**: remove variation from specified variables from the denominator. This computes the adjusted ICC with respect to the specified variables
- **adjustAll**: adjust for all variables. This computes the adjusted ICC with respect to all variables. This overrides the previous argument, so all variables are include in adjust.
- **showWarnings**: show warnings about model fit (default TRUE)
- **colinearityCutoff**: cutoff used to determine if model is computationally singular
- **control**: control settings for lmer()
- **nsim**: number of bootstrap datasets
- **...**: Additional arguments for lmer() or lm()
Details

A linear mixed model is fit for each gene, and bootMer() is used to generate parametric bootstrap confidence intervals. use.u=TRUE is used so that the \( \hat{u} \) values from the random effects are used as estimated and are not re-sampled. This gives confidence intervals as if additional data were generated from these same current samples. Conversely, use.u=FALSE assumes that this dataset is a sample from a larger population. Thus it simulates \( \hat{u} \) based on the estimated variance parameter. This approach gives confidence intervals as if additional data were collected from the larger population from which this dataset is sampled. Overall, use.u=TRUE gives smaller confidence intervals that are appropriate in this case.

Value

list() of where each entry is the result for a gene. Each entry is a matrix of the 95

Examples

```r
# load library
library(variancePartition)

# optional step to run analysis in parallel on multicore machines
# Here, we used 4 threads
library(doParallel)
cl <- makeCluster(4)
registerDoParallel(cl)
# or by using the doSNOW package

# load simulated data:
# geneExpr: matrix of gene expression values
# info: information/metadata about each sample
data(varPartData)

# Specify variables to consider
# Age is continuous so we model it as a fixed effect
# Individual and Tissue are both categorical, so we model them as random effects
form <- ~ Age + (1|Individual) + (1|Tissue)

# Compute bootstrap confidence intervals for each variable for each gene
resCI <- varPartConfInf( geneExpr[1:5,], form, info, nsim=100 )

# stop cluster
stopCluster(cl)
```

---

**Description**

A simulated dataset of gene expression and metadata

**Usage**

`data(varPartData)`
varPartData

Format

A dataset of 100 samples and 200 genes

Details

• geneCounts: gene expression in the form of RNA-seq counts
• geneExpr: gene expression on a continuous scale
• info: metadata about the study design

Value

varPartData
Index

*Topic datasets
  varPartData, 26

as.matrix
  (as.matrix, varPartResults-method), 2
  as.matrix, varPartResults-method, 2

calcVarPart, 3
  calcVarPart, lm-method (calcVarPart), 3
  calcVarPart, lmerMod-method (calcVarPart), 3

canCorPairs, 4

colinearityScore, 5

ESS, 6
  ESS, lmerMod-method (ESS), 6
  extractVarPart, 7

fitExtractVarPartModel, 8
  fitExtractVarPartModel, data.frame-method
    (fitExtractVarPartModel), 8
  fitExtractVarPartModel, EList-method
    (fitExtractVarPartModel), 8
  fitExtractVarPartModel, ExpressionSet-method
    (fitExtractVarPartModel), 8
  fitExtractVarPartModel, matrix-method
    (fitExtractVarPartModel), 8
  fitVarPartModel, 11
  fitVarPartModel, data.frame-method
    (fitVarPartModel), 11
  fitVarPartModel, EList-method
    (fitVarPartModel), 11
  fitVarPartModel, ExpressionSet-method
    (fitVarPartModel), 11
  fitVarPartModel, matrix-method
    (fitVarPartModel), 11

geneCounts (varPartData), 26
  geneExpr (varPartData), 26

getVarianceComponents, 13

ggColorHue, 14

info (varPartData), 26

plotCorrMatrix, 15
  plotCorrStructure, 16
  plotPercentBars, 17
  plotStratify, 18
  plotStratifyBy, 19
  plotVarPart, 21
  plotVarPart, data.frame-method
    (plotVarPart), 21
  plotVarPart, matrix-method
    (plotVarPart), 21
  plotVarPart, varPartResults-method
    (plotVarPart), 21

residuals
  (residuals, VarParFitList-method), 22
  residuals, VarParFitList-method, 22

sortCols, 23
  sortCols, data.frame-method (sortCols), 23
  sortCols, matrix-method (sortCols), 23
  sortCols, varPartResults-method
    (sortCols), 23

varPartConfInf, 25
  varPartData, 26