Package 'MethylIT'

March 21, 2019

Title Methylation Analysis Based on Information Thermodynamics

Version 0.3.1

Encoding UTF-8

Author Robersy Sanchez

Maintainer Robersy Sanchez <rus547@psu.edu>

Description Methylation Analysis based on Information Theory.

Depends R (>= 3.4.0),
DESeq2,
rtracklayer

License file LICENSE

biocViews Software, Epigenetics, MathematicalBiology, DNAMethylation, DifferentialMethylation, Sequencing, Alignment, Bayesian, DifferentialExpression

LazyData yes

Imports ArgumentCheck,
BiocGenerics,
BiocParallel,
Biostrings,
betareg,
biovizBase,
caret,
data.table,
Epi,
e1071,
genefilter,
GenomeInfoDb,
GenomicRanges,
graphics,
grDevices,
IRanges,
matrixStats,
MASS,
methods,

minpack.lm,
nls2,
parallel,
RCurl,
S4Vectors,
stats,
utils,
XML

RoxygenNote 6.1.0

Suggests testthat, rmarkdown,
knitr

VignetteBuilder rmarkdown,
knitr

R topics documented:

`AICmodel` *Akaike's Information Criterion (AIC)*

Description

this function permits the estimation of the AIC for models for which the function 'AIC' from the 'stats' package does not work.

Usage

```
AICmodel(model = NULL, residuals = NULL, np = NULL)
```

Arguments

`model`	if provided, it is an R object from where the residuals and model parameters can be retrieved using resid(model) and coef(model), respectively.
`residuals`	if provided, it is numerical vector with the residuals: residuals = observed.values - predicted.values, where predicted values are estimated from the model. If the parameter 'model' is not provided, then this parameter must be provided.
`np`	number of model parameters. If the parameter 'model' is not provided, then 'np' and 'residuals' must be provided.

Details

if for a given model 'm' AIC(m) works, then AICmodel(m) = AIC(m).

Value

AIC numerical value

Examples

```
set.seed(77)
x = runif(100, 1, 5)
y = 2 * exp(-0.5 * x) + runif(100, 0, 0.1)
plot(x, y)

nlm <- nls(Y ~ a * exp( b * X), data = data.frame(X=x, Y=y),
           start=list(a=1.5, b=-0.7),
           control=nls.control(maxiter=10^4, tol=1e-05),
```

```
            algorithm="port")
## The estimations of Akaike information criteria given by 'AIC' function
## from stats' R package and from 'AICmodel' function are equal.
AICmodel(nlm) == AIC(nlm)

## Now, using residuals from the fitted model:
res = y - coef(nlm)[1] * exp(coef(nlm)[2] * x)

AICmodel(residuals=res, np=2) == AIC(nlm)
```

`BICmodel` *Bayesian Information Criterion (BIC)*

Description

this function permits the estimation of the BIC for models for which the function 'BIC' from 'stats' packages does not work.

Usage

```
BICmodel(model = NULL, residuals = NULL, np = NULL)
```

Arguments

`model`	if provided, it is an R object from where the residuals and model parameters can be retrieved using resid(model) and coef(model), respectively.
`residuals`	if provided, it is numerical vector with the residuals: residuals = observe.values - predicted.values, where predicted values are estimated from the model. If the parameter 'model' is not provided, then this parameter must be provided.
`np`	number of model parameters. If the parameter 'model' is not provided, then 'np' and 'residuals' must be provided.

Details

if for a given model 'm' BIC(m) works, then BICmodel(m) = BIC(m).

Value

BIC numerical value

Examples

```
set.seed(77)
x = runif(100, 1, 5)
y = 2 * exp(-0.5 * x) + runif(100, 0, 0.1)
plot(x, y)
```

```
nlm <- nls(Y ~ a * exp( b * X), data = data.frame(X = x, Y = y),
           start = list( a = 1.5, b = -0.7),
           control = nls.control(maxiter = 10^4, tol = 1e-05),
           algorithm = "port")
## The estimations of Akaike information criteria given by BIC' function
## from stats' R package and from 'AICmodel' function are equals.
BICmodel(nlm) == BIC(nlm)

## Now, using residuals from the fitted model:
res = y - coef(nlm)[1] * exp(coef(nlm)[2] * x)

BICmodel(residuals = res, np = 2) == BIC(nlm)
```

---

countTest *Regression Test for Count*

Description

Perform Poisson and Negative Binomial regression analysis to compare the counts from different groups, treatment and control

Usage

```
countTest(DS, num.cores = 1, countFilter = TRUE, CountPerBp = NULL,
  minCountPerIndv = 3, maxGrpCV = NULL, FilterLog2FC = TRUE,
  pAdjustMethod = "BH", pvalCutOff = 0.05, MVrate = 0.98,
  Minlog2FC = 0.5, test = c("Wald", "LRT"), scaling = 1L,
  saveAll = FALSE, verbose = TRUE)
```

Arguments

DS	DESeqDataSet object
num.cores	number of cores used
countFilter	whether or not to filter the counts according to the minimum count per region per each individual/sample, which is setting by "minCountPerIndv"
CountPerBp	for each group the count per bp must be equal or greater than CountPerBp. The filter is applied if 'CountPerBp' is given and if 'x' DESeqDataSet object has the rowRanges as a GRanges object on it
minCountPerIndv	each gene or region must have more than 'minCountPerIndv' counts (on average) per individual in at least one group
maxGrpCV	A numerical vector. Maximum coefficient of variance for each group. Defaul maxGrpCV = NULL. The numbers maxGrpCV[1] and maxGrpCV[2] will be taken as the maximun variances values permitted in control and in treatment groups, repectively. If only maxGrpCV[1] is provided, then maxGrpCV = c(maxGrpCV[1], maxGrpCV[1]). This parameter is addressed to prevent testing

	regions where intra-group variations are very large, e.g.: control = c(1,0,1,1) and traatment = c(1, 0, 1, 40). The coefficient of variance for the treatment group is 1.87, very high. The generalized linear regression analysis would yield statistical significant group differences, but evidently there is something wrong in one of the treatment samples. We would try the application of further statistical smoothing approach, but we prefer to leave the user decide which regions to test.
`FilterLog2FC`	if TRUE, the results are filtered using the minimun absolute value of log2FoldChanges observed to accept that a gene in the treatment is differentially expressed in respect to the control
`pAdjustMethod`	method used to adjust the results; default: BH
`pvalCutOff`	cutoff used then a p-value adjustment is performed
`MVrate`	Minimum Mean/Variance rate
`Minlog2FC`	minimum logarithm base 2 of fold changes.
`test`	A character string matching one of "Wald" or "LRT". If test = "Wald", then the p-value of the Wald test for the coefficient of the independent variable (*treatment group*) will be reported p-value. If test = "LRT", then the p-value from a likelihood ratio test given by anova function from *stats* packages will be the reported p-value for the group comparison.
`scaling`	integer (default 1). Scaling factor estimate the signal density as: scaling * "DIMP-Count-Per-Bp". For example, if scaling = 1000, then signal density denotes the number of DIMPs in 1000 bp.
`saveAll`	if TRUE all the temporal results are returned
`verbose`	if TRUE, prints the function log to stdout

Value

a data frame or GRanges object (if the DS contain the GRanges information for each gene) with the test results and original count matrix, plus control and treatment signal densities and their variation.

Examples

```
countData <- matrix(1:40,ncol = 4)
condition <- factor(c("A","A","B","B"))
dds <- DESeqDataSetFromMatrix(countData, DataFrame(condition),
                              ~ condition)
countTest(dds)
```

`countTest2` *Regression Test for Count*

Description

Perform Poisson and Negative Binomial regression analysis to compare the counts from different groups, treatment and control. The difference between functions 'countTest2' and 'countTest' resides in the estimation of the prior weights used in Negative Binomial generalized linear model.

Usage

```
countTest2(DS, num.cores = 1, countFilter = TRUE, CountPerBp = NULL,
  minCountPerIndv = 3, maxGrpCV = NULL, FilterLog2FC = TRUE,
  pAdjustMethod = "BH", pvalCutOff = 0.05, MVrate = 0.98,
  Minlog2FC = 0.5, test = c("Wald", "LRT"), scaling = 1L,
  tasks = 0L, saveAll = FALSE, verbose = TRUE)
```

Arguments

DS	DESeqDataSet object
num.cores, tasks	Paramaters for parallele computation using package BiocParallel-package: the number of cores to use, i.e. at most how many child processes will be run simultaneously (see bplapply and the number of tasks per job (only for Linux OS).
countFilter	whether or not to filter the counts according to the minimum count per region per each individual/sample, which is setting by "minCountPerIndv".
CountPerBp	for each group the count per bp must be equal or greater than CountPerBp. The filter is applied if 'CountPerBp' is given and if 'x' DESeqDataSet object has the rowRanges as a GRanges object on it
minCountPerIndv	each gene or region must have more than 'minCountPerIndv' counts (on average) per individual in at least one group.
maxGrpCV	A numerical vector. Maximum coefficient of variance for each group. Defaul maxGrpCV = NULL. The numbers maxGrpCV[1] and maxGrpCV[2] will be taken as the maximun variances values permitted in control and in treatment groups, repectively. If only maxGrpCV[1] is provided, then maxGrpCV = c(maxGrpCV[1], maxGrpCV[1]). This parameter is addressed to prevent testing regions where intra-group variations are very large, e.g.: control = c(1,0,1,1) and traatment = c(1, 0, 1, 40). The coefficient of variance for the treatment group is 1.87, very high. The generalized linear regression analysis would yield statistical significant group differences, but evidently there is something wrong in one of the treatment samples. We would try the application of further statistical smoothing approach, but we prefer to leave the user decide which regions to test.
FilterLog2FC	if TRUE, the results are filtered using the minimun absolute value of log2FoldChanges observed to accept that a gene in the treatment is differentially expressed in respect to the control
pAdjustMethod	method used to adjust the results; default: BH
pvalCutOff	cutoff used then a p-value adjustment is performed
MVrate	Minimum Mean/Variance rate.
Minlog2FC	minimum logarithm base 2 of fold changes.
test	A character string matching one of "Wald" or "LRT". If test = "Wald", then the p-value of the Wald test for the coefficient of the independent variable (*treatment group*) will be reported p-value. If test = "LRT", then the p-value from a likelihood ratio test given by anova function from *stats* packages will be the reported p-value for the group comparison.

scaling	integer (default 1). Scaling factor estimate the signal density as: scaling * "DIMP-Count-Per-Bp". For example, if scaling = 1000, then signal density denotes the number of DIMPs in 1000 bp.
saveAll	if TRUE all the temporal results are returned
verbose	if TRUE, prints the function log to stdout

Value

a data frame or GRanges object (if the DS contain the GRanges information for each gene) with the test results and original count matrix, plus control and treatment signal densities and their variation.

Examples

```
countData <- matrix(1:40,ncol = 4)
condition <- factor(c("A","A","B","B"))
dds <- DESeqDataSetFromMatrix(countData, DataFrame(condition),
                              ~ condition)
countTest2(dds, verbose = FALSE)

# Parallel computation. Package 'BiocParallel' must be installed
# set.seed(133)
# countData <- matrix(sample.int(200, 10000, replace = TRUE), ncol = 4)
# condition <- factor(c("A","A","B","B"))
# dds <- DESeqDataSetFromMatrix(countData, DataFrame(condition),
                         ~ condition)
# countTest2(dds, num.cores = 4L)
```

---

estimateBayesianDivergence

Information divergence estimator

Description

The Information divergence of methylation levels is estimated using the direct estimation or a Bayessian approach of the methylation levels. Hellinger divergence is computed as given in reference 1.

Usage

```
estimateBayesianDivergence(x, Bayesian = FALSE, num.cores = 1,
  tasks = 0L, columns = c(mC1 = 1, uC1 = 2, mC2 = 3, uC2 = 4),
  meth.level = FALSE, preserve.gr = FALSE, verbose = TRUE)
```

Arguments

x	A matrix of counts or GRanges object with the table of counts in the meta-columns (methylated mC and unmethylated uC cytosines). Unless specified in the parameter 'columns', the methylation counts must be given in the first four columns: "mC1" and "uC1" methylated and unmethylated counts for control sample, and "mC2" and "uC2" methylated and unmethylated counts for treatment sample, respectively.
Bayesian	logical. Whether to perform the estimations based on posterior estimations of methylation levels.
num.cores	The number of cores to use, i.e. at most how many child processes will be run simultaneously (see 'bplapply' function from BiocParallel package).
tasks	integer(1). The number of tasks per job. value must be a scalar integer >= 0L. In this documentation a job is defined as a single call to a function, such as bplapply, bpmapply etc. A task is the division of the X argument into chunks. When tasks == 0 (default), X is divided as evenly as possible over the number of workers (see MulticoreParam from BiocParallel package).
columns	Vector of integer numbers of the columns where the counts "mC1", "uC1", "mC2", and "uC2" are in the matrix (default 1 to 4). That is, the input could have more than 4 columns, but only 4 columns with the counts are used.
meth.level	methylation levels can be provided in place of counts.
preserve.gr	Logic (Default:FALSE). Option of whether to preserve all the metadata from the original GRange object.
verbose	if TRUE, prints the function log to stdout

Details

For the current version, the Information divergence of methylation levels is estimated based on Hellinger divergence. If read counts are provided, then Hellinger divergence is computed as given in the first formula from Theorem 1 from reference 1. In the present case,

hdiv = 2*(n[1] + 1)*(n[2] + 1)*((sqrt(p[1]) - sqrt(p[2]))^2 + (sqrt(1 - p[1]) - sqrt(1 - p[2]))^2)/(n[1] + n[2] + 2)

where n[1] and n[2] are the coverage for the control and treatment, respectively. Notice that each row from the matrix of counts correspond to a single cytosine position and has four values corresponding to "mC1" and "uC1" (control), and mC2" and "uC2" for treatment.

If the methylation levels are provided in place of counts, then Hellinger divergence is computed as: hdiv = (sqrt(p[1]) - sqrt(p[2]))^2 + (sqrt(1 -p[1]) - sqrt(1 - p[2]))^2

This formula assumes that the probability vectors derived from the methylation levels (p_ij) p_j = c(p_ij, 1 - p_ij) (see function 'estimateHellingerDiv') are an unbiased estimation of the expected one.

Value

The input matrix or GRanges object with the four columns of counts and additional columns. If Bayessian = TRUE, the results are based on the posterior estimations of methylation levels. 1) p1" and "p2": methylation levels for control and treatment; 2) "hdiv": Hellinger divergence; 3) "bay.TV". "TV": total variation TV = p2 - p1 is based on simple counts

References

1. Basu A., Mandal A., Pardo L (2010) Hypothesis testing for two discrete populations based on the Hellinger distance. Stat Probab Lett 80: 206-214.

`estimateCutPoint` *Estimate cutpoints to distinguish the treatment methylation signal from the control*

Description

Given a list of two GRanges objects, control and treatment, carrying the potential signals (prior classification) from controls and treatments in terms of an information divergence (given the meta-columns), the function estimates the cutpoints of the control group versus treatment group.

Usage

```
estimateCutPoint(LR, control.names, treatment.names, simple = TRUE,
  column = c(hdiv = TRUE, TV = TRUE, wprob = FALSE, pos = FALSE),
  classifier1 = c("logistic", "pca.logistic", "lda", "qda", "pca.lda",
  "pca.qda"), classifier2 = NULL, tv.cut = 0.25, div.col = NULL,
  clas.perf = FALSE, post.cut = 0.5, prop = 0.6, n.pc = 1,
  find.cut = FALSE, cut.interval = c(0.5, 0.8), cut.incr = 0.01,
  stat = 1, num.cores = 1L, tasks = 0L, ...)
```

Arguments

`LR`	An object from 'pDMP' class. This obejct is previously obtained with function `getPotentialDIMP`.
`control.names, treatment.names`	Names/IDs of the control and treatment samples, which must be include in the variable LR.
`simple`	Logic (default, TRUE). If TRUE, then Youden Index is used to estimate the cutpoint.
`column`	a logical vector for column names for the predictor variables to be used: Hellinger divergence "hdiv", total variation "TV", probability of potential DIMP "wprob", and the relative cytosine site position "pos" in respect to the chromosome where it is located. The relative position is estimated as (x - x.min)/(x.max - x), where x.min and x.max are the maximum and minimum for the corresponding chromosome, repectively. If "wprob = TRUE", then Logarithm base-10 of "wprob" will be used as predictor in place of "wprob".
`classifier1, classifier2`	Classification model to use. Option "logistic" applies a logistic regression model; option "lda" applies a Linear Discriminant Analysis (LDA); "qda" applies a Quadratic Discriminant Analysis (QDA), "pca.logistic" applies logistic regression model using the Principal Component (PCs) estimated with Principal Component Analysis (PCA) as predictor variables. pca.lda" applies LDA using PCs

as predictor variables, and the option "pca.qda" applies a Quadratic Discriminant Analysis (QDA) using PCs as predictor variables. If classifier2 is not NULL, then it will be used to evaluate the classification performance, and the corresponding best fitted model will be returned.

`tv.cut` A cutoff for the total variation distance to be applied to each site/range. Only sites/ranges *k* with $TVD_k > tv.cut$ are are used in the analysis. Its value must be a number $0 < tv.cut < 1$. Default is $tv.cut = 0.25$.

`div.col` Column number for divergence variable for which the estimation of the cutpoint will be performed.

`clas.perf` Logic. Whether to evaluate the classification performance for the estimated cutpoint using a model classifier when 'simple=TRUE'. Default, FALSE.

`post.cut` If 'simple=FALSE', this is posterior probability to dicide whether a DMPs belong to treatment group. Default *post.cut* = 0.5.

`prop` Proportion to split the dataset used in the logistic regression: group versus divergence (at DIMPs) into two subsets, training and testing.

`n.pc` Number of principal components (PCs) to use if the classifier is not 'logistic'. In the current case, the maximun number of PCs is 4.

`find.cut` Logic. Wether to search for an optimal cutoff value to classify DMPs based on given specifications.

`cut.interval` 0 < *cut.interval* < 0.1. If *find.cut*= TRUE, the interval of treatment group posterior probabilities where to search for a cutpoint. Deafult *cut.interval* = c(0.5, 0.8).

`cut.incr` 0 < *cut.incr* < 0.1. If *find.cut*= TRUE, the sucesive increamental values runing on the interval *cut.interval*. Deafult, *cut.incr* = 0.01.

`stat` An integer number indicating the statistic to be used in the testing when *find.cut* = TRUE. The mapping for statistic names are: 0 = "Accuracy", 1 = "Sensitivity", 2 = "Specificity", 3 = "Pos Pred Value", 4 = "Neg Pred Value", 5 = "Precision", 6 = "Recall", 7 = "F1", 8 = "Prevalence", 9 = "Detection Rate", 10 = "Detection Prevalence", 11 = "Balanced Accuracy", 12 = FDR.

`num.cores, tasks`
Paramaters for parallele computation using package `BiocParallel-package`: the number of cores to use, i.e. at most how many child processes will be run simultaneously (see `bplapply` and the number of tasks per job (only for Linux OS).

Details

The function performs an estimation of the optimal cutpoint for the classification of the differentially methylated (cytosines) positions into two classes: DMPs from control and DMPs from treatment. The simplest approach to estimate the cutpoint is based on the application of Youden Index. More complexes approach based in several machine learning model are provided as well.

Results of the classification perfomance resulting from the estimated cutpoint are normally given, with the exception of those extreme situations where the statistics to evaluate performance cannot be estimated. More than one classifier model can be applied. For example, one classifier (logistic model) can be used to esitmate the posterior classification probabilities of DMP into those from

control and those from treatment. This probabilities are then used to estimate the cutpoint in range of values from, say, 0.5 to 0.8. Next, a different classifier can be used to evaluate the classification performance. Different classifier models would yield different performances. Models are returned and can be used in futher prediction with new datasets from the same batch experiment. This is a machine learnig approach to discriminate the biological regulatory signal naturally generated in the control from that one induced by the treatment.

Value

Depending the parameter setting will return the following list with elements:

1. cutpoint: Cutpoint estimated.
2. testSetPerformance: Performance evaluation on the test set.
3. testSetModel.FDR: False discovery rate on the test set.
4. model: Model used in the performance evaluation.
5. modelConfMatrix: Confusion matrix for the whole dataset derived applying the model classifier used in the performance evaluation.
6. initModel: Initial classifier model applied to estimate posterior classifications used in the cutpoint estimation.
7. postProbCut: Posterior probability used to estimate the cutpoint
8. classifier: Name of the model classifier used in the performance evaluation.
9. statistic: Name of the performance statistic used to find the cutpoint when find.cut = TRUE.
10. optStatVal: Value of the performance statistic at the cutpoint.

Examples

```
set.seed(123) ## To set a seed for random number generation
## GRanges object of the reference with methylation levels in
## its matacolumn
num.points <- 5000
Ref <- makeGRangesFromDataFrame(
  data.frame(chr = '1',
             start = 1:num.points,
             end = 1:num.points,
             strand = '*',
             p1 = rbeta(num.points, shape1 = 1, shape2 = 1.5)),
  keep.extra.columns = TRUE)

## List of Granges objects of individuals methylation levels
Indiv <- GRangesList(
  sample11 = makeGRangesFromDataFrame(
    data.frame(chr = '1',
               start = 1:num.points,
               end = 1:num.points,
               strand = '*',
               p2 = rbeta(num.points, shape1 = 1.5, shape2 = 2)),
    keep.extra.columns = TRUE),
  sample12 = makeGRangesFromDataFrame(
```

```
    data.frame(chr = '1',
               start = 1:num.points,
               end = 1:num.points,
               strand = '*',
               p2 = rbeta(num.points, shape1 = 1.6, shape2 = 2)),
    keep.extra.columns = TRUE),
  sample21 = makeGRangesFromDataFrame(
    data.frame(chr = '1',
               start = 1:num.points,
               end = 1:num.points,
               strand = '*',
               p2 = rbeta(num.points, shape1 = 40, shape2 = 4)),
    keep.extra.columns = TRUE),
  sample22 = makeGRangesFromDataFrame(
    data.frame(chr = '1',
               start = 1:num.points,
               end = 1:num.points,
               strand = '*',
               p2 = rbeta(num.points, shape1 = 41, shape2 = 4)),
    keep.extra.columns = TRUE))
## To estimate Hellinger divergence using only the methylation levels.
HD <- estimateDivergence(ref = Ref, indiv = Indiv, meth.level = TRUE,
                         columns = 1)
## To perform the nonlinear regression analysisx
nlms <- nonlinearFitDist(HD, column = 4, verbose = FALSE)

## Next, the potential signal can be estimated
PS <- getPotentialDIMP(LR = HD, nlms = nlms, div.col = 4, alpha = 0.05)

cutpoint <- estimateCutPoint(LR = PS, simple = TRUE, find.cut = FALSE,
                             column = c(hdiv = TRUE, TV = TRUE,
                                        wprob = TRUE, pos = TRUE),
                             interaction = "hdiv:TV", clas.perf = FALSE,
                             control.names = c("sample11", "sample12"),
                             treatment.names = c("sample21", "sample22"))
```

`estimateDivergence` *Information divergence estimator in respect to a reference sample*

Description

Wrapper of 'InfDiv' function to operate on list of GRanges

Usage

```
estimateDivergence(ref, indiv, Bayesian = FALSE, columns = NULL,
  min.coverage = 4, high.coverage = NULL, percentile = 0.999,
  num.cores = 1L, tasks = 0L, meth.level = FALSE, verbose = TRUE,
  ...)
```

Arguments

ref	The GRanges object of the reference individual that will be used in the estimation of the information divergence.
indiv	A list of GRanges objects from the individuals that will be used in the estimation of the information divergence.
Bayesian	Logical. Whether to perform the estimations based on posterior estimations of methylation levels.
columns	Vector of one or two integer numbers denoting the indexes of the columns where the methylated and unmethylated read counts are found or, if meth.level = TRUE, the columns corresponding to the methylation levels. If columns = NULL and meth.level = FALSE, then columns = c(1,2) is assumed. If columns = NULL and meth.level = TRUE, then columns = 1 is assumed.
min.coverage	Cytosine sites with coverage less than min.coverage are discarded.
high.coverage	An integer for read counts. Cytosine sites having higher coverage than this are discarded.
percentile	Threshold to remove the outliers from each file and all files stacked.
num.cores	The number of cores to use, i.e. at most how many child processes will be run simultaneously (see 'bplapply' function from BiocParallel package).
tasks	integer(1). The number of tasks per job. value must be a scalar integer >= 0L. In this documentation a job is defined as a single call to a function, such as bplapply, bpmapply etc. A task is the division of the X argument into chunks. When tasks == 0 (default), X is divided as evenly as possible over the number of workers (see MulticoreParam from BiocParallel package).
meth.level	Logic. Whether methylation levels are given in place of counts.
verbose	if TRUE, prints the function log to stdout
...	Additional parameters for 'uniqueGRanges' function.

Details

For the current version, the Information divergence of methylation levels is estimated based on Hellinger divergence. If read counts are provided, then Hellinger divergence is computed as given in the first formula from Theorem 1 from reference 1. In the present case: hdiv = 2*(n[1] + 1)*(n[2] + 1)*((sqrt(p[1]) - sqrt(p[2]))^2 + (sqrt(1-p[1]) - sqrt(1-p[2]))^2)/(n[1] + n[2] + 2)

where n[1] and n[2] are the coverage for the control and treatment, respectively. Notice that each row from the matrix of counts correspond to a single cytosine position and has four values corresponding to "mC1" and "uC1" (control), and mC2" and "uC2" for treatment.

If the methylation levels are provided in place of counts, then Hellinger divergence is computed as: hdiv = (sqrt(p[1]) - sqrt(p[2]))^2 + (sqrt(1 - p[1]) - sqrt(1 - p[2]))^2

This formula assumes that the probability vectors derived from the methylation levels (p_ij) p_j = c(p_ij, 1 - p_ij) (see function 'estimateHellingerDiv') are an unbiased estimation of the expected one. The function applies a pairwise filtering after building a single GRanges from the two GRanges objects. Experimentally available cytosine sites are paired using the function 'uniqueGRanges'.

Value

A list of GRanges objects with the four columns of counts, the information divergence, and additional columns: 1) The original matrix of methylated (c_i) and unmathylated (t_i) read counts from control (i=1) and treatment (i=2) samples. 2) p1" and "p2": methylation levels for control and treatment, respectively. 3) "bay.TV": total variation TV = p2 - p1. 4) "TV": total variation based on simple counts: TV=c1/(c1+t1)-c2/(c2+t2). 5) "hdiv": Hellinger divergence. If Bayesian = TRUE, the results are based on the posterior estimations of methylation levels.

Author(s)

Robersy Sanchez

Examples

```
num.samples <- 250
x <- data.frame(chr = "chr1", start = 1:num.samples,
                end = 1:num.samples,strand = '*',
                mC = rnbinom(size = num.samples, mu = 4, n = 500),
                uC = rnbinom(size = num.samples, mu = 4, n = 500))
y <- data.frame(chr = "chr1", start = 1:num.samples,
                end = 1:num.samples, strand = '*',
                mC = rnbinom(size = num.samples, mu = 4, n = 500),
                uC = rnbinom(size = num.samples, mu = 4, n = 500))
x <- makeGRangesFromDataFrame(x, keep.extra.columns = TRUE)
y <- makeGRangesFromDataFrame(y, keep.extra.columns = TRUE)
HD <- estimateDivergence(ref = x, indiv = list(y))
```

---

estimateECDF *A variant of Empirical Cumulative Distribution Function "ecdf"*

Description

This function is used to reduce the number of points used to build a ecdf of an experimental variable. When a variable has a very high amount of experimental values (several millions) could be computationally time consuming to perform a good-of-fit test and a non-linear regression estimation for a theoretical CDF based in such a big number of values.

Usage

```
estimateECDF(x, npoints = 10)
```

Arguments

x	numeric vector
npoints	minimum number of non-missing values

Details

The histogram cell midpoints values are used to build estimateECDF.

Value

ecdf of numeric vector

Examples

```
x <- sample(1:500, 50, replace=TRUE)
estimateECDF(x, npoints = 15)
```

`estimateHellingerDiv` *Hellinger divergence of methylation levels*

Description

Given a the methylation levels of two individual, the function computes the information divergence between methylation levels.

Usage

```
estimateHellingerDiv(p, n = NULL)
```

Arguments

p	A numerical vector of the methylation levels p = c(p1, p2) of individuals 1 and 2.
n	if supplied, it is a vector of integers denoting the coverages used in the estimation of the methylation levels.

Details

Each methylation level j for cytosine site i corresponds to a probability vector p_j = c(p_ij, 1 - p_ij). Then, the information divergence between methylation levels p1 and p2 is the divergence between the vectors p1 = c(p_i1, 1 - p_i1) and p2 = c(p_i2, 1 - p_i2). If the vector of covareges is supplied, then the information divergence is estimated according to the formula:

hdiv = 2*(n[1] + 1)*(n[2] + 1)*((sqrt(p[1]) - sqrt(p[2]))^2 + (sqrt(1 - p[1]) - sqrt(1 - p[2]))^2)/(n[1] + n[2] + 2) This formula corresponds to Hellinger divergence as given in the first formula from Theorem 1 from reference 1. Otherwise: hdiv = (sqrt(p[1]) - sqrt(p[2]))^2 + (sqrt(1 - p[1]) - sqrt(1 - p[2]))^2

Value

The Hellinger divergence value for the given methylation levels is returned

References

' 1. Basu A., Mandal A., Pardo L (2010) Hypothesis testing for two discrete populations based on the Hellinger distance. Stat Probab Lett 80: 206-214.

Examples

```
p <- c(0.5, 0.5)
estimateHellingerDiv(p)
```

`evaluateDIMPclass` *Evaluate DIMPs Classification*

Description

For a given cutpoint (previously estimated with the function estimateCutPoint), 'evaluateDIMPclass' will return the evaluation of the classification of DIMPs into two clases: DIMPS from control and DIMPs from treatment samples.

Usage

```
evaluateDIMPclass(LR, control.names, treatment.names, column = c(hdiv =
  FALSE, TV = FALSE, wprob = FALSE, pos = FALSE),
  classifier = c("logistic", "pca.logistic", "lda", "qda", "pca.lda",
  "pca.qda"), pval.col = NULL, n.pc = 1, center = FALSE,
  scale = FALSE, interaction = NULL, output = "conf.mat",
  prop = 0.6, num.boot = 100, num.cores = 1L, tasks = 0L,
  seed = 1234, verbose = TRUE)
```

Arguments

`LR`	A list of GRanges objects (LR) including control and treatment GRanges containing divergence values for each DIMP in the meta-column. LR is generated by the function 'selectDIMP' Each GRanges object must correspond to a sample. For example, if a sample is named 's1', then this sample can be accessed in the list of GRanges objects as LR$s1.
`control.names`	Names/IDs of the control samples, which must be include in thr variable LR.
`treatment.names`	Names/IDs of the treatment samples, which must be included in the variable LR.
`column`	a logical vector for column names for the predictor variables to be used: Hellinger divergence "hdiv", total variation "TV", probability of potential DIMP "wprob", and the relative cytosine site position "pos" in respect to the chromosome where it is located. The relative position is estimated as (x - x.min)/(x.max - x), where x.min and x.max are the maximum and minimum for the corresponding chromosome, repectively. If "wprob = TRUE", then Logarithm base-10 of "wprob" will be used as predictor in place of "wprob".

classifier	Classification model to use. Option "logistic" applies a logistic regression model; option "lda" applies a Linear Discriminant Analysis (LDA); "qda" applies a Quadratic Discriminant Analysis (QDA), "pca.logistic" applies logistic regression model using the Principal Component (PCs) estimated with Principal Component Analysis (PCA) as predictor variables. pca.lda" applies LDA using PCs as predictor variables, and the option "pca.qda" applies a Quadratic Discriminant Analysis (QDA) using PCs as predictor variables.
pval.col	Column number for p-value used in the performance analysis and estimation of the cutpoints. Default: NULL. If NULL it is assumed that the column is named "wprob".
n.pc	Number of principal components (PCs) to use if the classifier is not 'logistic'. In the current case, the maximun number of PCs is 4.
center	A logical value indicating whether the variables should be shifted to be zero centered (same as in 'prcomp' prcomp). Only used if classifier = "pcaLDA".
scale	A logical value indicating whether the variables should be scaled to have unit variance before the analysis takes place (same as in 'prcomp' prcomp). Only used if classifier = "pcaLDA".
interaction	Variable interactions to consider in a logistic regression model. Any pairwise combination of the variable "hdiv", "TV", "wprob", and "pos" can be provided. For example: "hdiv:TV", "wprob:pos", "wprob:TV", etc.
output	Type of output to request: output = c("conf.mat", "mc.val", "boot.all", "all"). See below.
prop	Proportion to split the dataset used in the logistic regression: group versus divergence (at DIMPs) into two subsets, training and testing.
num.boot	Number of bootstrap validations to perform in the evaluation of the logistic regression: group versus divergence (at DIMPs).
num.cores, tasks	Paramaters for parallele computation using package BiocParallel-package: the number of cores to use, i.e. at most how many child processes will be run simultaneously (see bplapply and the number of tasks per job (only for Linux OS).
seed	Random seed used for random number generation.
verbose	if TRUE, prints the function log to stdout

Details

The regulatory methylation signal is also an output from a natural process that continuously takes place across the ontogenetic development of the organisms. So, we expect to see methylation signal on natural ordinary conditions. Here, to distinguish a control methylation signal from a treatment, three classification models are provided: 1) logistic, 2) Linear Discriminant Analysis (LDA) and 3) Quadratic Discriminant Analysis (QDA). In particular, four predictor variables can be used: Hellinger divergence "hdiv", total variation "TV", probability of potential DIMP "wprob" and DIMP genomic coordinated "pos". Principal component analysis (PCA) is used to convert a set of observations of possibly correlated predictor variables into a set of values of linearly uncorrelated variables (principal components, PCs). The PCs are used as new, uncorrelated predictor variables for LDA, QDA, and logistic classifiers.

A classification result with low accuracy and compromising values from other classification performance indicators (see below) suggest that the treatment does not induce a significant regulatory signal different from control.

Value

output = "conf.mat" will perform a logistic regression group versus divergence (at DIMPs) to evaluate the discrimination between control-DIMPs and treatment-DIMPs. The evaluation of this classification is provided through the function 'confusionMatrix' from R package 'caret'. "mc.val" will perform a 'num.boot'-times Monte Carlo (bootstrap) validation and return a summary. By default function 'confusionMatrix' from R package caret' randomly splits the sample into two subsets, training and testing, according to the supplied proportion 'prop' (i.e., prop = 0.6). After selecting output = "mc.val", the function 'confusionMatrix' will be executed 'num.boot'-times, each time performing a different random split of the sample. "boot.all" same as "mc.val" plus a matrix with statistcs reported by 'confusionMatrix'. "all" return a list with all the mentioned outputs.

Examples

```
set.seed(123) ## To set a seed for random number generation
## GRanges object of the reference with methylation levels in
## its matacolumn
num.points <- 5000
Ref <- makeGRangesFromDataFrame(
    data.frame(chr = '1',
            start = 1:num.points,
            end = 1:num.points,
            strand = '*',
            p1 = rbeta(num.points, shape1 = 1, shape2 = 1.5)),
    keep.extra.columns = TRUE)

## List of Granges objects of individuals methylation levels
Indiv <- GRangesList(
    sample11 = makeGRangesFromDataFrame(
        data.frame(chr = '1',
            start = 1:num.points,
            end = 1:num.points,
            strand = '*',
            p2 = rbeta(num.points, shape1 = 1.5, shape2 = 2)),
         keep.extra.columns = TRUE),
    sample12 = makeGRangesFromDataFrame(
        data.frame(chr = '1',
            start = 1:num.points,
            end = 1:num.points,
            strand = '*',
            p2 = rbeta(num.points, shape1 = 1.6, shape2 = 2)),
        keep.extra.columns = TRUE),
    sample21 = makeGRangesFromDataFrame(
        data.frame(chr = '1',
            start = 1:num.points,
            end = 1:num.points,
            strand = '*',
            p2 = rbeta(num.points, shape1 = 40, shape2 = 4)),
```

```
        keep.extra.columns = TRUE),
    sample22 = makeGRangesFromDataFrame(
        data.frame(chr = '1',
            start = 1:num.points,
            end = 1:num.points,
            strand = '*',
            p2 = rbeta(num.points, shape1 = 41, shape2 = 4)),
        keep.extra.columns = TRUE))
## To estimate Hellinger divergence using only the methylation levels.
HD <- estimateDivergence(ref = Ref, indiv = Indiv, meth.level = TRUE,
                         columns = 1)
## To perform the nonlinear regression analysis
nlms <- nonlinearFitDist(HD, column = 4, verbose = FALSE)

## Next, the potential signal can be estimated
PS <- getPotentialDIMP(LR = HD, nlms = nlms, div.col = 4, alpha = 0.05)

## The cutpoint estimation used to discriminate the signal from the noise
cutpoints <- estimateCutPoint(PS, control.names = c("sample11", "sample12"),
                         treatment.names = c("sample21", "sample22"),
                         div.col = 4, verbose = TRUE)
## DIMPs are selected using the cupoints
DIMPs <- selectDIMP(PS, div.col = 4, cutpoint = min(cutpoints$cutpoint))

## Classification of DIMPs into two clases: DIMPS from control and DIMPs from
## treatment samples and evaluation of the classifier performance (for more
## details see ?evaluateDIMPclass).
conf.mat <- evaluateDIMPclass(DIMPs,
                         column = c(hdiv = TRUE, TV = TRUE,
                         wprob = TRUE, pos = FALSE),
                         control.names = c("sample11", "sample12"),
                         treatment.names = c("sample21", "sample22"))
confusion.matrix <- conf.mat$conf.mat
model.fit <- summary(conf.mat$model)
```

`filterByCoverage` *Filter methylation counts by coverage*

Description

The function is used to discard the cytosine positions with coverage values less than 'min.coverage' read counts or values greater than the specified 'percentile'.

Usage

```
filterByCoverage(x, min.coverage = 4, max.coverage = Inf,
  percentile = 0.999, col.names = c(coverage = NULL, mC = NULL, uC =
  NULL), verbose = TRUE)
```

Arguments

x	GRanges object or list of GRanges
min.coverage	Cytosine sites with coverage less than min.coverage are discarded. Default: 0
max.coverage	Cytosine sites with coverage greater than max.coverage are discarded. Default: Inf
percentile	Threshold to remove the outliers from each file and all files stacked. If percentile is 1, all the outliers stay
col.names	The number of the 'coverage' column. Since no specific table format for the count data is specified, at least the number of the 'coverage' column must be given, or the number of the columns with methylated (mC) and unmethylated counts (uC). Then coverage = mC + uC.
verbose	If TRUE, prints the function log to stdout

Details

The input must be a GRanges object or list of GRanges objects with a coverage column in the meta-column table or the columns with methylated (mC) and unmethylated counts (uC).

Value

The input GRanges object or list of GRanges objects after filtering them.

Examples

```
gr1 <- makeGRangesFromDataFrame(
    data.frame(chr = "chr1", start = 11:15, end = 11:15,
            strand = c("+","-","+","*","."), mC = 1, uC = 1:5),
    keep.extra.columns = TRUE)
 filterByCoverage(gr1, min.coverage = 1, max.coverage = 4,
            col.names = c(mC = 1, uC = 2), verbose = FALSE)
```

---

filterGRange *Filter methylation counts by coverage in a GRange object*

Description

The function is used to discard the cytosine positions with coverage values less than 'min.coverage' read counts or values greater than the specified 'percentile'.

Usage

```
filterGRange(x, min.coverage = 4, max.coverage = Inf,
  percentile = 0.999, col.names = c(coverage = NULL, mC = NULL, uC =
  NULL), sample.name = "", verbose = TRUE)
```

Arguments

x	GRanges object
min.coverage	Cytosine sites with coverage less than min.coverage are discarded. Default: 0
max.coverage	Cytosine sites with coverage greater than max.coverage are discarded. Default: Inf
percentile	Threshold to remove the outliers from each file and all files stacked. If percentile is 1, all the outliers stay
col.names	The number of the 'coverage' column. Since no specific table format for the count data is specified, at least the number of the 'coverage' column must be given, or the number of the columns with methylated (mC) and unmethylated counts (uC). Then coverage = mC + uC.
sample.name	Name of the sample
verbose	If TRUE, prints the function log to stdout

Details

The input must be a GRanges object with a coverage column in the meta-olumn table or the columns with methylated (mC) and unmethylated counts (uC).

Value

The input GRanges object or list of GRanges objects after filtering it.

Examples

```
gr1 <- makeGRangesFromDataFrame(
    data.frame(chr = "chr1", start = 11:15, end = 11:15,
            strand = c("+","-","+","*","."), mC = 1, uC = 1:5),
    keep.extra.columns = TRUE)
filterGRange(gr1, min.coverage = 1, max.coverage = 4,
            col.names = c(mC = 1, uC = 2), verbose = FALSE)
```

FisherTest *Fisher's exact test for read counts on GRanges objects*

Description

Given a GRanges object with the methylated and unmethylated read counts for control and treatment in its metacolumn, Fisher's exact test is performed for each cytosine site.

Usage

```
FisherTest(LR, count.col = 1:2, control.names = NULL,
  treatment.names = NULL, pooling.stat = "sum", tv.cut = NULL,
  hdiv.cut = NULL, hdiv.col = NULL, pAdjustMethod = "BH",
  pvalCutOff = 0.05, saveAll = FALSE, num.cores = 1L, tasks = 0L,
  verbose = FALSE, ...)
```

Arguments

LR	A list of GRanges, a GRangesList, a CompressedGRangesList object. Each GRanges object from the list must have two columns: methylated (mC) and unmethylated (uC) counts. The name of each element from the list must coincide with a control or a treatment name.
count.col	2d-vector of integers with the indexes of the read count columns. If not given, then it is asssumed that the methylated and unmethylated read counts are located in columns 1 and 2 of each GRanges metacolumns. If object LR is the output of Methyl-IT function estimateDivergence, then columns 1:4 are the read count columns: columns 1 and 2 are methylated and unmethylated read counts from the reference group, while columns 3 and 4 are methylated and unmethylated read counts from the treatment group, respectively. In this case, if the requested comparison is reference versus treatment, then no specification is needed for count.col. The comparison control versus treatment can be obtained by setting count.col = 3:4 and providing control.names and treatment.names.
control.names, treatment.names	Names/IDs of control and treatment samples, which must be included in the variable GR at the metacolumn. Default NULL. If provided the Fisher's exact test control versus trearment is performed. Default is NULL. If NULL, then it is assumed that each GRanges object in LR has four columns of counts. The first two columns correspond to the methylated and unmethylated counts from control/reference and the other two columns are the methylated and unmethylated counts from treatment, respectively.
pooling.stat	statistic used to estimate the methylation pool: row sum, row mean or row median of methylated and unmethylated read counts across individuals. If the number of control samples is greater than 2 and pooling.stat is not NULL, then they will pooled. The same for treatment. Otherwise, all the pairwise comparisons will be done.
tv.cut	A cutoff for the total variation distance (TVD; absolute value of methylation levels differences) estimated at each site/range as the difference of the group means of methylation levels. If tv.cut is provided, then sites/ranges k with abs(TV_k) < tv.cut are removed before performing the regression analysis. Its value must be NULL or a number 0 < tv.cut < 1.
hdiv.cut	An optional cutoff for the Hellinger divergence (*hdiv*). If the LR object derives from the previous application of function estimateDivergence, then a column with the *hdiv* values is provided. If combined with tv.cut, this permits a more effective filtering of the signal from the noise. Default is NULL.
hdiv.col	Optional. Columns where *hdiv* values are located in each GRange object from LR. It must be provided if together with *hdiv.cut*. Default is NULL.
pAdjustMethod	method used to adjust the results; default: BH
pvalCutOff	cutoff used then a p-value adjustment is performed
saveAll	if TRUE all the temporal results are returned
num.cores	The number of cores to use, i.e. at most how many child processes will be run simultaneously (see bpapply function from BiocParallel).

tasks	integer(1). The number of tasks per job. value must be a scalar integer >= 0L. In this documentation a job is defined as a single call to a function, such as bplapply, bpmapply etc. A task is the division of the X argument into chunks. When tasks == 0 (default), X is divided as evenly as possible over the number of workers (see MulticoreParam from BiocParallel package).
verbose	if TRUE, prints the function log to stdout
...	Additional parameters for function MethylIT.

Details

Samples from each group are pooled according to the statistic selected (see parameter pooling.stat) and a unique GRanges object is created with the methylated and unmethylated read counts for each group (control and treatment) in the metacolumn. So, a contingency table can be built for range from GRanges object.

Value

The input GRanges object with the columns of Fisher's exact test p-value, total variation (difference of methylation levels), and p-value adjusment.

See Also

rmstGR

Examples

```
#' A list of GRanges
set.seed(123)
sites = 15
data <- list(
  C1 = data.frame(chr = "chr1", start = 1:sites,
                  end = 1:sites,strand = '*',
                  mC = rnbinom(size = 8, mu = 3, n = sites),
                  uC = rnbinom(size = 50, mu = 10, n = sites)),
  C2 = data.frame(chr = "chr1", start = 1:sites,
                  end = 1:sites, strand = '*',
                  mC = rnbinom(size = 8, mu = 3, n = sites),
                  uC = rnbinom(size = 50, mu = 10, n = sites)),
  T1 = data.frame(chr = "chr1", start = 1:sites,
                  end = 1:sites,strand = '*',
                  mC = rnbinom(size = 50, mu = 10, n = sites),
                  uC = rnbinom(size = 10, mu = 10, n = sites)),
  T2 = data.frame(chr = "chr1", start = 1:sites,
                  end = 1:sites, strand = '*',
                  mC = rnbinom(size = 50, mu = 10, n = sites),
                  uC = rnbinom(size = 5, mu = 10, n = sites)))
#' Transforming the list of data frames into a list of GRanges objects
data = lapply(data,
              function(x)
                makeGRangesFromDataFrame(x, keep.extra.columns = TRUE))
```

```
FisherTest(LR = data, control.names = c("C1", "C2"),
        treatment.names = c("T1", "T2"), tv.cut = 0.25,
        pAdjustMethod="BH", pvalCutOff = 0.05, num.cores = 1L,
        verbose=TRUE)
```

`fitGammaDist` *Nonlinear fit of Gamma CDF (Gamma)*

Description

This function performs the nonlinear fit of GGamma CDF of a variable x

Usage

```
fitGammaDist(x, probability.x, parameter.values, location.par = FALSE,
  summarized.data = FALSE, sample.size = 20, npoints = NULL,
  maxiter = 1024, ftol = 1e-12, ptol = 1e-12, maxfev = 1e+05,
  verbose = TRUE)
```

Arguments

`x`	numerical vector
`probability.x`	probability vector of x. If not provided, the values are estimated using the empirical cumulative distribution function ('ecdf') from 'stats' R package.
`parameter.values`	initial parameter values for the nonlinear fit. If the locator paramter is included (mu != 0), this must be given as parameter.values = list(alpha = 'value', scale = 'value', mu = 'value') or if mu = 0, as: parameter.values = list(alpha = 'value', scale = 'value'). If not provided, then an initial guess is provided.
`location.par`	whether to consider the fitting to generalized gamma distribution (Gamma) including the location parameter, i.e., a Gamma with four parameters (GGamam3P).
`summarized.data`	Logic value. If TRUE (default: FALSE), summarized data based on 'npoints' are used to perform the nonlinear fit.
`sample.size`	size of the sample.
`npoints`	number of points used in the fit.
`maxiter`	positive integer. Termination occurs when the number of iterations reaches maxiter. Default value: 1024.
`ftol`	non-negative numeric. Termination occurs when both the actual and predicted relative reductions in the sum of squares are at most ftol. Therefore, ftol measures the relative error desired in the sum of squares. Default value: 1e-12
`ptol`	non-negative numeric. Termination occurs when the relative error between two consecutive iterates is at most ptol. Therefore, ptol measures the relative error desired in the approximate solution. Default value: 1e-12.

maxfev	integer; termination occurs when the number of calls to fn has reached maxfev. Note that nls.lm sets the value of maxfev to 100*(length(par) + 1) if maxfev = integer(), where par is the list or vector of parameters to be optimized.
verbose	if TRUE, prints the function log to stdout

Details

The algorithm tries to fit the two-parameter Gamma CDF ("Gamma2P") or the three-parameter Gamma ("Gamma3P") using a modification of Levenberg-Marquardt algorithm implemented in function 'nls.lm' from 'minpack.lm' package that is used to perform the nonlinear fit. Cross-validations for the nonlinear regressions (R.Cross.val) were performed in each methylome as described in reference [1]. In addition, Stein's formula for adjusted R squared (rho) was used as an estimator of the average cross-validation predictive power [1].

If the number of values to fit is >10^6, the fitting to a GGamma CDF would be a time consuming task. To reduce the computational time, the option 'summarized.data' can be set 'TRUE'. If summarized.data = TRUE, the original variable values are summarized into 'npoint' bins and their midpoints are used as the new predictors. In this case, only the goodness-of-fit indicators AIC and R.Cross.val are estimated based on all the original variable x values.

Value

Model table with coefficients and goodness-of-fit results: Adj.R.Square, deviance, AIC, R.Cross.val, and rho, as well as, the coefficient covariance matrix.

Author(s)

Robersy Sanchez - 06/03/2016

References

1. Stevens JP. Applied Multivariate Statistics for the Social Sciences. Fifth Edit. Routledge Academic; 2009.

Examples

```
set.seed(126)
x <- rgamma(1000, alpha = 1.03, scale = 2.1)
fitGammaDist(x)
```

---

fitGGammaDist *Nonlinear fit of Generalized Gamma CDF (GGamma)*

Description

This function performs the nonlinear fit of GGamma CDF of a variable x

Usage

```
fitGGammaDist(x, probability.x, parameter.values, location.par = FALSE,
  summarized.data = FALSE, sample.size = 20, npoints = NULL,
  maxiter = 1024, ftol = 1e-12, ptol = 1e-12, maxfev = 1e+05,
  verbose = TRUE)
```

Arguments

x	numerical vector
probability.x	probability vector of x. If not provided, the values are estimated using the empirical cumulative distribution function ('ecdf') from 'stats' R package.
parameter.values	initial parameter values for the nonlinear fit. If the locator paramter is included (mu != 0), this must be given as parameter.values = list(alpha = 'value', scale = 'value', mu = 'value', psi = 'value') or if mu = 0, as: parameter.values =list(alpha = 'value', scale = 'value', psi = 'value'). If not provided, then an initial guess is provided.
location.par	whether to consider the fitting to generalized gamma distribution (GGamma) including the location parameter, i.e., a GGamma with four parameters (GGamam4P).
summarized.data	Logic value. If TRUE (default: FALSE), summarized data based on 'npoints' are used to perform the nonlinear fit.
sample.size	size of the sample.
npoints	number of points used in the fit.
maxiter	positive integer. Termination occurs when the number of iterations reaches maxiter. Default value: 1024.
ftol	non-negative numeric. Termination occurs when both the actual and predicted relative reductions in the sum of squares are at most ftol. Therefore, ftol measures the relative error desired in the sum of squares. Default value: 1e-12
ptol	non-negative numeric. Termination occurs when the relative error between two consecutive iterates is at most ptol. Therefore, ptol measures the relative error desired in the approximate solution. Default value: 1e-12.
maxfev	integer; termination occurs when the number of calls to fn has reached maxfev. Note that nls.lm sets the value of maxfev to 100*(length(par) + 1) if maxfev = integer(), where par is the list or vector of parameters to be optimized.
verbose	if TRUE, prints the function log to stdout

Details

The script algorithm tries to fit the three-parameter GGamma CDF ("GGamma3P") or the four-parameter GGamma ("GGamma4P") using a modification of Levenberg-Marquardt algorithm implemented in function 'nls.lm' from 'minpack.lm' package that is used to perform the nonlinear fit. Cross-validations for the nonlinear regressions (R.Cross.val) were performed in each methylome as described in reference [1]. In addition, Stein's formula for adjusted R squared (rho) was used as an estimator of the average cross-validation predictive power [1].

If the number of values to fit is >10^6, the fitting to a GGamma CDF would be a time consuming task. To reduce the computational time, the option summarized.data' can be set 'TRUE'. If summarized.data = TRUE, the original variable values are summarized into 'npoint' bins and their midpoints are used as the new predictors. In this case, only the goodness-of-fit indicators AIC and R.Cross.val are estimated based on all the original variable x values.

Value

Model table with coefficients and goodness-of-fit results: Adj.R.Square, deviance, AIC, R.Cross.val, and rho, as well as, the coefficient covariance matrix.

Author(s)

Robersy Sanchez - 06/03/2016

References

1. Stevens JP. Applied Multivariate Statistics for the Social Sciences. Fifth Edit. Routledge Academic; 2009.

Examples

```
set.seed(126)
x <- rggamma(1000, alpha = 1.03, psi = 0.75, scale = 2.1)
fitGGammaDist(x)
```

`getDIMPatGenes` *Count DIMPs at gene-body*

Description

The function counts DIMPs overlapping with gene-body. In fact, this function also can be used to count DIMPs overlapping with any set of regions given in a GRanges object.

Usage

```
getDIMPatGenes(GR, ...)

## Default S3 method:
getDIMPatGenes(GR, GENES, ignore.strand = TRUE)

## S3 method for class 'pDMP'
getDIMPatGenes(GR, GENES, ignore.strand = TRUE)

## S3 method for class 'InfDiv'
getDIMPatGenes(GR, GENES, ignore.strand = TRUE)

## S3 method for class 'list'
getDIMPatGenes(GR, GENES, ignore.strand = TRUE)
```

Arguments

GR	An obects object from the any of the classes: 'pDMP', 'InfDiv', GRangesList, GRanges or a list of GRanges.
GENES	A GRanges object with gene coordinates and gene IDs. A meta-column named 'gene_id' carying the gene ids should be included. If the meta-column named 'gene_id' is not provided, then gene (region) ids will be created using the gene (region) coordinates.
ignore.strand	When set to TRUE, the strand information is ignored in the calculations. Default value: TRUE

Value

A GRanges object

Examples

```
num.points <- 10000 # Number of cytosine position with methylation call
## Gene annotation
genes <- GRanges(seqnames = "1",
                ranges = IRanges(start = c(3631, 6788, 11649),
                                 end = c(5899, 9130, 13714)),
                strand = c("+", "-", "-"))
mcols(genes) <- data.frame(gene_id = c("AT1G01010", "AT1G01020",
                                       "AT1G01030"))

set.seed(123) ## To set a seed for random number generation
## GRanges object of the reference with methylation levels in
## its meta-column
Ref <- makeGRangesFromDataFrame(
    data.frame(chr = '1',
                start = 1:num.points,
                end = 1:num.points,
                strand = '*',
                p2 = rbeta(num.points, shape1 = 1, shape2 = 1.5)),
    keep.extra.columns = TRUE)

## List of Granges objects of individuals methylation levels
Indiv <- GRangesList(
    sample11 = makeGRangesFromDataFrame(
        data.frame(chr = '1',
                start = 1:num.points,
                end = 1:num.points,
                strand = '*',
                p2 = rbeta(num.points, shape1 = 1.5, shape2 = 2)),
        keep.extra.columns = TRUE),
    sample12 = makeGRangesFromDataFrame(
        data.frame(chr = '1',
                start = 1:num.points,
                end = 1:num.points,
                strand = '*',
                p2 = rbeta(num.points, shape1 = 1.6, shape2 = 2.1)),
```

```
        keep.extra.columns = TRUE),
    sample21 = makeGRangesFromDataFrame(
        data.frame(chr = '1',
                start = 1:num.points,
                end = 1:num.points,
                strand = '*',
                p2 = rbeta(num.points, shape1 = 10, shape2 = 4)),
        keep.extra.columns = TRUE),
    sample22 = makeGRangesFromDataFrame(
        data.frame(chr = '1',
                start = 1:num.points,
                end = 1:num.points,
                strand = '*',
                p2 = rbeta(num.points, shape1 = 11, shape2 = 4)),
        keep.extra.columns = TRUE))
## To estimate Hellinger divergence using only the methylation levels.
HD <- estimateDivergence(ref = Ref, indiv = Indiv, meth.level = TRUE,
                         columns = 1)
## To perform the nonlinear regression analysis
nlms <- nonlinearFitDist(HD, column = 4, verbose = FALSE)

## Next, the potential signal can be estimated
PS <- getPotentialDIMP(LR = HD, nlms = nlms, div.col = 4, alpha = 0.05)

## The cutpoint estimation used to discriminate the signal from the noise
cutpoints <- estimateCutPoint(PS, control.names = c("sample11", "sample12"),
                              treatment.names = c("sample21", "sample22"),
                              div.col = 4, verbose = TRUE)
## DIMPs are selected using the cupoints
DIMPs <- selectDIMP(PS, div.col = 4, cutpoint = min(cutpoints$cutpoint))

## Finally DIMPs found on genes
DIMR <- getDIMPatGenes(GR = DIMPs$sample21, GENES = genes)
```

`getGEOSuppFiles` *Get Supplemental Files from GEO*

Description

Decompress 'gzip' files.

Usage

```
getGEOSuppFiles(GEO, makeDirectory = FALSE, baseDir = getwd(),
  pattern = NULL, verbose = TRUE)
```

Arguments

GEO	A character vector with GEO accession numbers.
makeDirectory	Logic (FALSE). If GEO accession number is provided, whether to create a subdirectory for the downloaded files.
baseDir	Directory where files are downloads if GEO accession number is provided. Default is the current working directory.
pattern	A pattern for the name of the supplementary files from the GEO dataset. If provided, then only the files with the given pattern are downloaded. Otherwise, all the supplementary files are downloaded.
verbose	If TRUE, prints the function log to stdout

Details

Download supplemental files from a specified GEO dataset. This function is originally provided in the Bioconductor package 'GEOquery'. The original function download all the supplemental files for a given GEO accession number. Herein small detail is added to permit only the download of the specified files and from several GEO accession numbers with only one call to the function.

Value

A data frame is returned invisibly with rownames representing the full path of the resulting downloaded files and the records in the data.frame the output of file.info for each downloaded file.

Author(s)

Original author: Sean Davis <sdavis2@mail.nih.gov>

Examples

```
## Download supplementary files from GEO data set and store "fullpath/name"
## in variable filename. The parameter 'pattern' permits us to download only
## the specified filesCG, in this case, CG and CHG methylation contexts.

filenames <- getGEOSuppFiles(GEO = "GSM881757",
                pattern = "G_cytosine.txt.gz")

## Read the files with function 'readCounts2GRangesList'. Only lines starting
## with the word 'Chr1' will be read, in acccordance with the  specification
## given with parameter 'pattern'

LR <- readCounts2GRangesList(filenames = filenames, remove = TRUE,
                             sample.id = c("drm2_CG", "drm2_CHG"),
                             columns = c(seqnames = 1, start = 2, mC = 4,
                                     uC = 3),
                             pattern = "^Chr1", verbose = TRUE)
file.remove(filenames) ## Remove the downloaded file
```

`getPotentialDIMP` *Potential methylation signal*

Description

This function perform a selection of the cytosine sites carrying the potential methylation signal. The potential signals from controls and treatments are used as prior classification in further step of signal detection.

Usage

```
getPotentialDIMP(LR, nlms = NULL, div.col, dist.name = "Weibull2P",
  absolute = FALSE, alpha = 0.05, tv.col = NULL, tv.cut = NULL,
  min.coverage = NULL, hdiv.col = NULL, hdiv.cut = NULL)
```

Arguments

`LR`	An object from 'InfDiv' class. This obejct is previously obtained with function `estimateDivergence`.
`nlms`	A list of distribution fitted models (output of 'fitNonlinearWeibullDist' function) or NULL. If NULL, then empirical cumulative distribution function is used to get the potential DIMPs.
`div.col`	Column number for divergence variable is located in the meta-column.
`dist.name`	name of the distribution to fit: Weibull2P (default: "Weibull2P"), Weibull three-parameters (Weibull3P), gamma with three-parameter (Gamma3P), gamma with two-parameter (Gamma2P), generalized gamma with three-parameter ("GGamma3P") or four-parameter ("GGamma4P"), the empirical cumulative distribution function (ECDF) or "None".
`absolute`	Logic (default, FALSE). Total variation (TV, the difference of methylation levels) is normally an output in the downstream MethylIT analysis. If 'absolute = TRUE', then TV is transformed into |TV|, which is an information divergence that can be fitted to Weibull or to Generalized Gamma distribution. So, if the nonlinear fit was performed for |TV|, then absolute must be set to TRUE.
`alpha`	A numerical value (usually alpha < 0.05) used to select cytosine sites k with information divergence (DIV_k) for which Weibull probability P[DIV_k > DIV(alpha)].
`tv.col`	Column number for the total variation to be used for filtering cytosine positions (if provided).
`tv.cut`	If tv.cut and tv.col are provided, then cytosine sites k with abs(TV_k) < tv.cut are removed before to perform the ROC analysis.
`min.coverage`	Cytosine sites with coverage less than min.coverage are discarded. Default: 0
`hdiv.col`	Optional. A column number for the Hellinger distance to be used for filtering cytosine positions. Fedault is NULL.
`hdiv.cut`	If hdiv.cut and hdiv.col are provided, then cytosine sites k with hdiv < hdiv.cut are removed.

Details

The potential signals are cytosine sites k with information divergence (DIV_k) values greater than the DIV(alpha = 0.05). The value of alpha can be specified. For example, potential signals with DIV_k > DIV(alpha = 0.01) can be selected. For each sample, cytosine sites are selected based on the corresponding fitted Weilbull distribution model that has been supplied.

Value

A list of GRanges objects, each GRanges object carrying the selected cytosine sites and and the Weibull probability P[DIV_k > DIV(alpha)].

Examples

```
num.points <- 1000
HD <- GRangesList( sample1 = makeGRangesFromDataFrame(
        data.frame(chr = "chr1", start = 1:num.points, end = 1:num.points,
            strand = '*',
            hdiv = rweibull(1:num.points, shape = 0.75, scale = 1)),
        keep.extra.columns = TRUE))
nlms <- nonlinearFitDist(HD, column = 1, verbose = FALSE)
getPotentialDIMP(LR = HD, nlms = nlms, div.col = 1, alpha = 0.05)
```

---

ggamma *Generalized Gamma distribution*

Description

Probability density function (PDF), cummulative density function (CDF), quantile function and random generation for the Generalized Gamma (GG) distribution with 3 or 4 parameters: alpha, scale, mu, and psi. The function is reduced to GGamma distribution with 3 parameters by setting mu = 0.

Usage

```
dggamma(q, alpha = 1, scale = 1, mu = 0, psi = 1, log.p = FALSE)

pggamma(q, alpha = 1, scale = 1, mu = 0, psi = 1,
  lower.tail = TRUE, log.p = FALSE)

qggamma(p, alpha = 1, scale = 1, mu = 0, psi = 1,
  lower.tail = TRUE, log.p = FALSE)

rggamma(n, alpha = 1, scale = 1, mu = 0, psi = 1)
```

Arguments

q	numeric vector
alpha	numerical parameter, strictly positive (default 1). The generalized gamma becomes the gamma distribution for alpha = 1.
scale, psi	the same real positive parameters as is used for the Gamma distribution. These are numerical and strictly positives; default 1. (see ?pgamma).
mu	location parameter (numerical, default 0).
log.p	logical; if TRUE, probabilities/densities p are returned as log(p).
lower.tail	logical; if TRUE (default), probabilities are P[X<=x], otherwise, P[X > x]
n	number of observations

Details

Details about these function can be found in references 1 to 3. You may also see section Note at ?pgamma or ?rgamma. Herein, we are using Stacy' s formula (references 2 to 3) with the parametrization given in reference 4 (equation 6, page 12). As in the case of gamma distribution function, the cumulative distribution function (as given in equation 12, page 13 from reference 4) is expressed in terms of the lower incomplete gamma function (see ?pgamma).

The GG distribution with parameters α, β (scale), ψ, and μ has density:

$$f(x|\alpha,\beta,\mu,\psi) = \alpha exp(-((x-\mu)/\beta)^{\alpha})((x-\mu)/\beta)^{(}\alpha*\psi-1)/(\beta Gamma(\psi))$$

Value

GG PDF values (3-parameters or 4-parameters) for dggamma, GG probability for pggamma, quaniles or GG random generated values for rggamma.

References

1. Handbook on STATISTICAL DISTRIBUTIONS for experimentalists (p. 73) by Christian Walck. Particle Physics Group Fysikum. University of Stockholm (e-mail: walck@physto.se)

2. Stacy, E. W. A Generalization of the Gamma Distribution. Ann. Math. Stat. 33, 1187–1192 (1962).

3. Stacy E, Mihram G (1965) Parameter estimation for a generalized gamma distribution. Technometrics 7: 349-358.

4. Sanchez, R. & Mackenzie, S. A. Information Thermodynamics of Cytosine DNA Methylation. PLoS One 11, e0150427 (2016).

Examples

```
q <- (1:9)/10
pggamma(q, alpha = 1, scale = 1, mu = 0,
        psi = 1, lower.tail = TRUE, log.p = FALSE)

## To fit random generated numbers
set.seed(126)
```

```
x <- rggamma(1000, alpha = 1.03, psi = 0.75, scale = 2.1)
fitGGammaDist(x)
```

`lapply` *Apply a function over a list-like object preserving its attributes*

Description

lapply returns a list of the same length as 'x', each element of which is the result of applying FUN to the corresponding element of 'x'.

Usage

```
lapply(x, FUN, ...)

## Default S3 method:
lapply(x, FUN, keep.attr = FALSE, ...)
```

Arguments

`x`	A list-like or vector-like object
`FUN, ...`	See ?base::lapply for a description of these arguments.
`keep.attr`	Logic. If TRUE, then the original attributes from 'x' are preserved in the returned list. Derfaul is FALSE.

Value

Same as in ?base::lapply if keep.attr = FALSE. Otherwise same values preserving original attributes from 'x'.

See Also

base::lapply

Examples

```
# Create a list
x <- list(a = 1:10, beta = exp(-3:3), logic = c(TRUE,FALSE,FALSE,TRUE))
class(x) <- "nice"

# compute the list mean for each list element using 'base::lapply'
class(lapply(x, mean))

# To preserve attributes
class(lapply(x, mean, keep.attr = TRUE))
```

`MethylIT` *MethylIT: A package for methylation analysis*

Description

This package helps to do methylation analysis based on information thermodynamics and signal detection

`nonlinearFitDist` *Nonlinear fit of Information divergences distribution*

Description

A wrapper to call functions 'Weibull3P' and 'fitGGammaDist' to operate on list of GRanges.

Usage

```
nonlinearFitDist(LR, column = 9, dist.name = "Weibull",
  sample.size = 20, location.par = FALSE, absolute = FALSE,
  npoints = NULL, npoints0 = NULL, summarized.data = FALSE,
  maxiter = 1024, tol = 1e-12, ftol = 1e-12, ptol = 1e-12,
  minFactor = 10^-6, num.cores = NULL, tasks = 0L, maxfev = 1e+05,
  verbose = TRUE)
```

Arguments

`LR`	A list of GRanges objects with information divergence values in their meta-columns.
`column`	An integer number denoting the index of the GRanges column where the information divergence is given. Default column = 1
`dist.name`	name of the distribution to fit: Weibull (default: "Weibull"), gamma with three-parameter (Gamma3P), gamma with two-parameter (Gamma2P), generalized gamma with three-parameter ("GGamma3P") or four-parameter ("GGamma4P").
`sample.size`	size of the sample
`location.par`	whether to consider the fitting to generalized gamma distribution (GGamma) including the location parameter, i.e., a GGamma with four parameters (GGamam4P).
`absolute`	Logic (default, FALSE). Total variation (TV, the difference of methylation levels) is normally an output in the downstream MethylIT analysis. If 'absolute = TRUE', then TV is transformed into |TV|, which is an information divergence that can be fitted to Weibull or to Generalized Gamma distribution.
`npoints`	number of points used in the fit

npoints0	subset of points where to estimate the ECDF (used only to reduce computational time)
summarized.data	Logic value. If TRUE (default: FALSE), summarized data based on 'npoints' are used to perform the nonlinear fit. Only for GGamma distribution.
maxiter	positive integer. Termination occurs when the number of iterations reaches maxiter. Default value: 1024
tol	A positive numeric value specifying the tolerance level for the relative offset convergence criterion. Default value: 1e-12,
ftol	non-negative numeric. Termination occurs when both the actual and predicted relative reductions in the sum of squares are at most ftol. Therefore, ftol measures the relative error desired in the sum of squares. Default value: 1e-12
ptol	non-negative numeric. Termination occurs when the relative error between two consecutive iterates is at most ptol. Therefore, ptol measures the relative error desired in the approximate solution. Default value: 1e-12,
minFactor	A positive numeric value specifying the minimum step-size factor allowed on any step in the iteration. The increment is calculated with a Gauss-Newton algorithm and successively halved until the residual sum of squares has been decreased or until the step-size factor has been reduced below this limit. Default value: 10^-6.
num.cores	The number of cores to use, i.e. at most how many child processes will be run simultaneously (see bplapply function from BiocParallel package).
tasks	integer(1). The number of tasks per job. value must be a scalar integer >= 0L. In this documentation a job is defined as a single call to a function, such as bplapply, bpmapply etc. A task is the division of the X argument into chunks. When tasks == 0 (default), X is divided as evenly as possible over the number of workers (see MulticoreParam from BiocParallel package).
maxfev	integer; termination occurs when the number of calls to fn has reached maxfev. Note that nls.lm sets the value of maxfev to 100*(length(par) + 1) if maxfev = integer(), where par is the list or vector of parameters to be optimized.
verbose	If TRUE, prints the function log to stdout
...	other parameters

Details

The algorithm prepares the information divergence variable to try fitting Weibull or generalized gamma distribution model to the data. If Weibull distribution is selected (default: "Weibull"), function 'Weibull2P' first attempts fitting to the two-parameter Weibull CDF (Weibull2P). If Weibull2P did not fit, then the algorithm will try to fit Weibull3P. The Levenberg-Marquardt algorithm implemented in R package 'minpack.lm' is used to perform the nonlinear fit. Cross-validations for the nonlinear regressions (R.Cross.val) are performed in each methylome as described in reference [1]. In addition, Stein's formula for adjusted R squared (rho) is used as an estimator of the average cross-validation predictive power [1].

If "GGamma3P" is selected the call to function 'fitGGammaDist' permits the fitting to the three-parameter GGamma CDF ("GGamma3P"). The fit to the four-parameter GGamma ("GGamma4P")

is also available. GGamma distribution are fitted using a modification of Levenberg-Marquardt algorithm implemented in function 'nls.lm' from the 'minpack.lm' R package. Notice that the fit to GGamma dsitribution is computationally time consuming (see ?fitGGammaDist for additional information).

Value

Model table with coeficients and goodness-of-fit results: Adj.R.Square, deviance, AIC, R.Cross.val, and rho, as well as, the coefficient covariance matrix.

Author(s)

Robersy Sanchez 01/31/2018

References

1. Stevens JP. Applied Multivariate Statistics for the Social Sciences. Fifth Edit. Routledge Academic; 2009. 2. [1] R. Sanchez and S. A. Mackenzie, "Information Thermodynamics of Cytosine DNA Methylation," PLoS One, vol. 11, no. 3, p. e0150427, Mar. 2016.

Examples

```
## The Weilbull distribution is a particular case of GGamma.
## The goodness-of-fit indicators AIC, BIC and R.Cross.val suggest that the
## best fit randomly generated values with Weibull distribution is obtained
## using the Weibull model (in this example).
set.seed(123)
num.points <- 1000
HD <- GRangesList(
    sample1 <- makeGRangesFromDataFrame(
        data.frame(chr = "chr1", start = 1:num.points, end = 1:num.points,
            strand = '*',
            hdiv = rweibull(1:num.points, shape = 0.75, scale = 1)),
        keep.extra.columns = TRUE))
nlms <- nonlinearFitDist(HD, column = 1, verbose = FALSE)
nlms2 <- nonlinearFitDist(HD, column = 1, dist.name = "GGamma3P",
        verbose = FALSE)

## We used the parameter values estimated for "GGamma3P" in the last
## example (nlms2) to generate random values with GGamma disitribution. The
## goodness-of-fit indicators AIC, BIC and R.Cross.val suggest that the
## best fit is obtained for GGamma model.
num.points <- 1000
HD <- GRangesList(
    sample1 <- makeGRangesFromDataFrame(
        data.frame(chr = "chr1", start = 1:num.points, end = 1:num.points,
            strand = '*',
            hdiv = rggamma(num.points, alpha = 0.75, psi = 1.02,
                scale = 0.97)),
        keep.extra.columns = TRUE))
nlms3 <- nonlinearFitDist(HD, column = 1, verbose = FALSE)
nlms4 <- nonlinearFitDist(HD, column = 1, dist.name = "GGamma3P",
```

```
          verbose = FALSE)
```

---

pcaLDA *Linear Discriminant Analysis (LDA) using Principal Component Analysis (PCA)*

Description

The principal components (PCs) for predictor variables provided as input data are estimated and then the individual coordinates in the selected PCs are used as predictors in the LDA

Predict using a PCA-LDA model built with function 'pcaLDA'

Usage

```
pcaLDA(formula = NULL, data = NULL, grouping = NULL, n.pc = 1,
              scale = FALSE, center = FALSE, tol = 1.0e-4, method = "moment",
               max.pc = NULL)

## S3 method for class 'pcaLDA'
predict(object, newdata, type = c("lda.pred", "class",
  "posterior", "scores", "pca.ind.coord"), ...)
```

Arguments

formula	Same as in 'lda'from pakage 'MASS'.
data	Same as in 'lda'from pakage 'MASS'.
grouping	Same as in 'lda' from pakage 'MASS'.
n.pc	Number of principal components to use in the LDA.
scale	Same as in 'prcomp' from pakage 'prcomp'.
center	Same as in 'prcomp' from pakage 'prcomp'.
tol	Same as in 'prcomp' from pakage 'prcomp'.
method	Same as in 'lda'from pakage 'MASS'.
max.pc	Same as in paramter 'rank.' from pakage 'prcomp'.
object	To use with function 'predict'. A 'pcaLDA' object containing a list of two objects: 1) an object of class inheriting from "lda" and 2) an object of class inheriting from "prcomp".
newdata	To use with function 'predict'. New data for classification prediction
type	To use with function 'predict'. . The type of prediction required. The default is "all" given by function 'predict.lda' from MASS package: 'class', 'posterior', and 'scores' (see ?predict.lda).
...	Not in use.

Details

The principal components (PCs) are obtained using the function 'prcomp' from R pacakage 'stats', while the LDA is performed using the 'lda' function from R package 'MASS'. The current application only use basic functionalities of mentioned functions. As shown in the example, pcaLDA' function can be used in general classification problems.

Value

Function 'pcaLDA' returns an object ('pcaLDA' class) consisting of list with two objects: 1) 'lda': an object of class 'lda' from package 'MASS'. 2) 'pca': an object of class 'prcomp' from package 'stats'. For information on how to use these objects see ?lda and ?prcomp.

Examples

```
data(iris)
ld1 <- pcaLDA(formula = Species ~ Petal.Length + Sepal.Length + Sepal.Width,
            data = iris, n.pc = 1, max.pc = 2, scale = TRUE, center = TRUE)
## === Prediction === ##
ld2 <- pcaLDA(formula = Species ~., data = iris, n.pc = 1, max.pc = 2,
             scale = TRUE, center = TRUE)
set.seed(123)
idx <- sample.int(150, 40)
newdata <- iris[idx, 1:4]
newdata.prediction <- predict(ld2, newdata = newdata)

## The confusion matrix
x <- data.frame(TRUE.class = iris$Species[idx],
               PRED.class = newdata.prediction$class)
table(x)
```

---

pcaLogisticR *Linear Discriminant Analysis (logistic) using Principal Component Analysis (PCA)*

Description

The principal components (PCs) for predictor variables provided as input data are estimated and then the individual coordinates in the selected PCs are used as predictors in the logistic

Logistic regresion using Principal Components from PCA as predictor variables

Usage

```
pcaLogisticR(formula = NULL, data = NULL, n.pc = 1, scale = FALSE,
  center = FALSE, tol = 1e-04, max.pc = NULL)

## S3 method for class 'pcaLogisticR'
predict(object, newdata, type = c("class",
  "response", "pca.ind.coord", "all"), ...)
```

Arguments

formula	Same as in 'glm' from pakage 'stats'.
data	Same as in 'glm' from pakage 'stats'.
n.pc	Number of principal components to use in the logistic.
scale	Same as in 'prcomp' from pakage 'prcomp'.
center	Same as in 'prcomp' from pakage 'prcomp'.
tol	Same as in 'prcomp' from pakage 'prcomp'.
max.pc	Same as in paramter 'rank.' from pakage 'prcomp'.
object	To use with function 'predict'. A 'pcaLogisticR' object containing a list of two objects: 1) an object of class inheriting from "glm" and 2) an object of class inheriting from "prcomp".
newdata	To use with function 'predict'. New data for classification prediction
type	To use with function 'predict'. The type of prediction required: "class", "response", "pca.ind.coord", or "all". If type = 'all', function 'predict.pcaLogisticR' ('predict') returns a list with: 1) 'class': individual classification. 2) 'response': probabilities for the positive class. 3) 'pca.ind.coord': PC individual coordinate. Each element of this list can be requested independently using parameter 'type'.
...	Not in use.

Details

The principal components (PCs) are obtained using the function 'prcomp' from R pacakage 'stats', while the logistic is performed using the 'logistic' function from R package 'MASS'. The current application only use basic functionalities of mentioned functions. As shown in the example, 'pcaLogisticR' function can be used in general classification problems.

Value

Function 'pcaLogisticR' returns an object ('pcaLogisticR' class) containing a list of two objects:

1. 'logistic': an object of class 'glm' from package 'stats'.
2. 'pca': an object of class 'prcomp' from package 'stats'.
3. reference.level: response level used as reference.
4. positive.level: response level that corresponds to a "positive" result. When type = "response", the probability vector returned correspond to the probabilities of each individual to be a result, i.e., the probability to belong to the class of positive level.

For information on how to use these objects see ?glm and ?prcomp.

Examples

```
data(iris)
data <- iris[ iris$Species != "virginica", ]
data$Species <- droplevels(data$Species)
formula <- Species ~ Petal.Length + Sepal.Length + Petal.Width
pca.logistic <- pcaLogisticR(formula = formula,
```

```
                          data = data, n.pc = 2, scale = TRUE,
                          center = TRUE, max.pc = 2)
set.seed(123)
newdata <- iris[sample.int(150, 40), 1:4]
newdata.prediction <- predict(pca.logistic, newdata, type = "all")
```

---

pcaQDA *Quadratic Discriminant Analysis (QDA) using Principal Component Analysis (PCA)*

Description

The principal components (PCs) for predictor variables provided as input data are estimated and then the individual coordinates in the selected PCs are used as predictors in the qda

Predict using a PCA-LDA model built with function 'pcaLDA'

Usage

```
pcaQDA(formula = NULL, data = NULL, grouping = NULL, n.pc = 1,
  scale = FALSE, center = FALSE, tol = 1e-04, method = "moment",
  max.pc = NULL)

## S3 method for class 'pcaQDA'
predict(object, newdata, type = c("qda.pred", "class",
  "posterior", "pca.ind.coord", "all"), ...)
```

Arguments

formula	Same as in 'qda'from pakage 'MASS'.
data	Same as in 'qda'from pakage 'MASS'.
grouping	Same as in 'qda'from pakage 'MASS'.
n.pc	Number of principal components to use in the qda.
scale	Same as in 'prcomp' from pakage 'prcomp'.
center	Same as in 'prcomp' from pakage 'prcomp'.
tol	Same as in 'prcomp' from pakage 'prcomp'v.
method	Same as in 'qda'from pakage 'MASS'.
max.pc	Same as in paramter 'rank.' from prcomp' prcomp.
object	To use with function 'predict'. A 'pcaQDA' object containing a list of two objects: 1) an object of class inheriting from "qda" and 2) an object of class inheriting from "prcomp".
newdata	To use with function 'predict'. New data for classification prediction.
type	To use with function 'predict'. The type of prediction required. The default is "all" given by function 'predict.qda' from MASS package: 'class', 'posterior', and 'scores' (see ?predict.QDA).
...	Not in use.

Details

The principal components (PCs) are obtained using the function 'prcomp' from R pacakage 'stats', while the qda is performed using the 'qda' function from R package 'MASS'. The current application only uses basic functionalities of mentioned functions. As shown in the example, 'pcaQDA' function can be used in general classification problems.

Value

Function 'pcaQDA' returns an object ("pcaQDA") consisting of a list with two objects: 1) 'qda': an object of class 'qda' from package 'MASS'. 2) 'pca': an object of class 'prcomp' from package 'stats'. For information on how to use these objects see ?qda and ?prcomp.

Examples

```
data(iris)
qd1 <- pcaQDA(formula = Species ~ Petal.Length + Sepal.Length + Sepal.Width,
            data = iris, n.pc = 1, max.pc = 2, scale = TRUE, center = TRUE)
## === Prediction === ##
qd2 <- pcaQDA(formula = Species ~., data = iris, n.pc = 1, max.pc = 2,
             scale = TRUE, center = TRUE)
set.seed(123)
idx <- sample.int(150, 40)
newdata <- iris[idx, 1:4]
newdata.prediction <- predict(qd2, newdata = newdata, type = "all")

## The confusion matrix
x <- data.frame(TRUE.class = iris$Species[idx],
               PRED.class = newdata.prediction$class)
table(x)
```

---

poolFromGRlist — *Methylation pool from a list of GRanges objects with methylation read counts*

Description

This function will build a GRanges methylation pool from a list of GRanges objects

Usage

```
poolFromGRlist(LR, stat = "sum", num.cores = 1, tasks = 0L,
  prob = FALSE, column = 1L, verbose = TRUE, ...)
```

Arguments

LR	list of GRanges objects to build a virtual individual (methylation pool)
stat	statistic used to estimate the methylation pool: row sum, row mean or row median of methylated and unmethylated read counts across individuals

num.cores	The number of cores to use, i.e. at most how many child processes will be run simultaneously (see bplapply function from BiocParallel package).
tasks	integer(1). The number of tasks per job. Value must be a scalar integer >= 0L. In this documentation a job is defined as a single call to a function, such as bplapply, bpmapply etc. A task is the division of the X argument into chunks. When tasks == 0 (default), X is divided as evenly as possible over the number of workers (see MulticoreParam from BiocParallel package).
prob	Logic. Whether the variable for pooling is between 0 and 1 (a probability), e.g., methylation levels. If TRUE, then Fisher's transformation is applied, the row mean is computed for each cytosine site and returned in the original measurement scale between 0 and 1 by using the inverse of Fisher's transformation.
column	If prob == TRUE, then the 'column' from the LR metacolumns where the prob values are found must be provided. Otherwise, column = 1L.
verbose	If TRUE, prints the function log to stdout
...	Additional parameters for 'uniqueGRanges' function.

Details

The list of GRanges objects (LR) provided to build a virtual methylome should be an output of the function 'readCounts2GRangesList' or at least each GRanges must have the columns named "mC" and "uC", for the read counts of methylated and unmethylated cytosines, respectively.

Value

A GRanges object

Examples

```
gr1 <- makeGRangesFromDataFrame(
    data.frame(chr = "chr1", start = 11:15, end = 11:15,
               strand = '*', mC = 1, uC = 1:5),
    keep.extra.columns = TRUE)
gr2 <- makeGRangesFromDataFrame(
    data.frame(chr = "chr1", start = 11:15, end = 11:15,
               strand = '*', mC = 1, uC = 1:5),
    keep.extra.columns = TRUE)

answer <- poolFromGRlist(list(gr1, gr2), stat = 'sum', verbose = FALSE)
```

---

predict.LogisticR *Predict function for logistic regression model from 'LogisticR' class*

Description

Predict using a logistic model obtained from the output of function evaluateDIMPclass.

Usage

```
## S3 method for class 'LogisticR'
predict(object, newdata = NULL, type = c("class",
  "posteriors", "all"), ...)
```

Arguments

object	To use with function 'predict'. An object from 'LogisticR' class. A logistic model given by function evaluateDIMPclass.
newdata	To use with function 'predict'. New data for classification prediction. Optionally, a data frame in which to look for variables with which to predict. If omitted, the fitted linear predictors are used.
type	To use with function 'predict'. . The type of prediction required. The default is "class". Possible outputs are: 'class', 'posterior', and 'scores' (see ?predict.lda).
...	Not in use.

Details

This function is specific for predictions based on a logistic model given by function evaluateDIMPclass. A logistic model is obtained with 'glm' regression can be used directly with function 'predict' from 'stats' package.

Value

A character vector of prediction classes or a numeric vector of probabilities or a list containing the two vectors: prediction classes and 'posteriors' probabilities.

---

print.CutPoint *Printing object from 'CutPoint' class by simple print methods*

Description

Printing object from 'CutPoint' class by simple print methods

Usage

```
## S3 method for class 'CutPoint'
print(x, digits = getOption("digits"))
```

Arguments

x	Object of class "CutPoint"
digits	Number of significant digits to be used.

pweibull3P *Weibull distribution with three parameters*

Description

Density, distribution function, quantile function and random generation for the Weibull distribution with three parameters

Usage

```
pweibull3P(q, shape = 1, scale = 1, mu = 0)
```

Arguments

q	vector of quantiles
shape	shape parameter, or slope, defaulting to 1
scale	scale parameter, or characteristic life, defaulting to 1
mu	location parameter, or failure free life, defaulting to 0

Value

3 parameters Weibull distribution

Examples

```
num.samples <- 10000
shape <- 0.75
scale <- 1
x <- rweibull(num.samples, shape = shape, scale = scale)
wei.model <- weibull3P(x)
y <- pweibull3P(x,
                shape = as.numeric(wei.model$Estimate[1]),
                scale = as.numeric(wei.model$Estimate[2]),
                mu = as.numeric(wei.model$Estimate[3]) )
```

readCounts2GRangesList *Read files of methylation count tables*

Description

This function is addressed to read files with methylation count table data commonly generated after the alignment of BS-seq data or found in GEO database

Usage

```
readCounts2GRangesList(filenames = NULL, sample.id = NULL,
  pattern = NULL, remove = FALSE, columns = c(seqnames = NULL, start
  = NULL, end = NULL, strand = NULL, fraction = NULL, percent = NULL, mC =
  NULL, uC = NULL, coverage = NULL, context = NULL),
  chromosome.names = NULL, chromosomes = NULL, verbose = TRUE, ...)
```

Arguments

filenames	Character vector with the file names
sample.id	Character vector with the names of the samples corresponding to each file
pattern	Chromosome name pattern. Users working on Linux OS can specify the reading of specific lines from each file by using regular expressions.
remove	Logic (TRUE). Usually the supplementary files from GEO datasets are 'gz' compressed. File datasets must be decompressed to be read. The decompressed files are removed after read if this is set 'TRUE'.
columns	Vector of integer numbers denoting the table columns that must be read. The numbers for 'seqnames' (chromosomes), 'start', and 'end' (if different from 'start') columns must be given. The possible fields are: 'seqnames' (chromosomes),'start', 'end', 'strand', 'fraction', percent' (metylation percentage), 'mC' (methylates cytosine), 'uC' (non methylated cytosine), 'coverage', and 'context' (methylation context). These column headers are not required to be in the files.
chromosome.names	If provided, for each GRanges object, chromosome names will be changed to those provided in 'chromosome.names' applying seqlevels(x) <- chromosome.names'. This option permits to use all the functionality of the function "seqlevels" defined from package "GenomeInfoDb", which rename, add, and reorder the seqlevels all at once (see ?seqlevels).
chromosomes	If provided, it must be a character vector with the names of the chromosomes that you want to include in the final GRanges objects.
verbose	If TRUE, prints the function log to stdout
...	Additional parameters for 'fread' function from 'data.table' package

Details

Read tables from files with a table methylation count data using the function fread from the package 'data.table' and and yields a list of GRanges objects with the information provided.

Value

A list of GRanges objects

Examples

```
## Create a cov file with it's file name including "gz" (tarball extension)
filename <- "./file.cov"
gr1 <- data.frame(chr = c("chr1", "chr1"), post = c(1,2),
```

```
                strand = c("+", "-"), ratio = c(0.9, 0.5),
                context = c("CG", "CG"), CT = c(20, 30))
filename <- "./file.cov"
write.table(as.data.frame(gr1), file = filename,
            col.names = TRUE, row.names = FALSE, quote = FALSE)

## Read the file. It does not work. Typing mistake: "fractions"
LR <- try(readCounts2GRangesList(filenames = filename, remove = FALSE,
                            sample.id = "test",
                            columns = c(seqnames = 1, start = 2,
                                    strand = 3, fractions = 4,
                                    context = 5, coverage = 6)),
                                    silent = TRUE)
file.remove(filename) # Remove the file

## Read the file
## Create a cov file with it's file name including "gz" (tarball extension)
filename <- "./file.cov"
gr1 <- data.frame(chr = c("chr1", "chr1"), post = c(1,2),
                strand = c("+", "-"), ratio = c(0.9, 0.5),
                context = c("CG", "CG"), CT = c(20, 30))
filename <- "./file.cov"
write.table(as.data.frame(gr1), file = filename,
            col.names = TRUE, row.names = FALSE, quote = FALSE)

LR <- readCounts2GRangesList(filenames = filename, remove = TRUE,
                             sample.id = "test",
                             columns = c(seqnames = 1, start = 2,
                                         strand = 3, fraction = 4,
                                         context = 5, coverage = 6))

## Download supplementary files from GEO data set and store "fullpath/name"
## in variable filename. The parameter 'pattern' permits us to download only
## the specified filesCG, in this case, CG and CHG methylation contexts.

filenames <- getGEOSuppFiles(GEO = "GSM881757",
                            pattern = "G_cytosine.txt.gz")

## Read the files with function 'readCounts2GRangesList'. Only lines starting
## with the word 'Chr1' will be read, in acccordance with the  specification
##given with parameter 'pattern'

LR <- readCounts2GRangesList(filenames = filenames, remove = TRUE,
                            sample.id = c("drm2_CG", "drm2_CHG"),
                            columns = c(seqnames = 1, start = 2,
                                        mC = 4, uC = 3),
                            pattern = "^Chr1", verbose = TRUE)
file.remove(filenames) # Remove the downloaded file
```

---

selectDIMP *Selection of DIMPs*

Description

For a given cutpoint (previously estimated with the function estimateCutPoint), 'selectDIMP' will return the differentially informative methyated positions (DIMPs). DIMPs are cytosine positions for which the divergence is greater than the cutpoint.

Usage

```
selectDIMP(LR, div.col = NULL, pval.col = NULL, absolute = FALSE,
  cutpoint, tv.col = NULL, tv.cut = NULL)
```

Arguments

LR	A list of GRanges objects including control and treatment GRanges. Each GRanges object must correspond to a sample. For example, if a sample is named 's1', then this sample can be accessed in the list of GRanges objects as LR$s1.
div.col	Number of the column where the divergence variable (i.e., Hellinger divergence) is located in the GRanges meta-columns.
pval.col	If the cutpoints is a p-value, then the column number for p-values should be provided. Default: NULL. Notice that one of the parameter values div.col or pval.col must be given.
absolute	Logic (default, FALSE). Total variation (TV, the difference of methylation levels) is normally an output in the downstream MethylIT analysis. If 'absolute = TRUE', then TV is tranformed into |TV|, which is an information divergence that can be fitted to Weibull or to Generalized Gamma distribution. So, if the nonlinear fit was performed for |TV|, then here absolute must be set to TRUE.
cutpoint	Cutpoint to select DIMPs. Cytosine positions with divergence greater than 'cutpoint' will selected as DIMPs. Cutpoints are estimated with the function 'estimateCutPoint'.
tv.col	Column number for the total variation to be used for filtering cytosine positions (if provided).
tv.cut	If tv.cut and tv.col are provided, then cytosine sites k with abs(TV_k) < tv.cut are removed.

Details

Theoretically a DIMP denotes a cytosine position with high probability to be differentially methylated. That is, in the statistical molecular-biophysics context, a DIMP must be considered only in a probabilistic term and not as an absolute deterministic experimental output.

The uncertainty and dynamics of the DNA methylation process, the continuous action of the omnipresent thermal fluctuations, as well as, the inherent stochasticity of the biochemical reactions make it impossible to ensure whether a specific cytosine position is methylated in an absolutely deterministic sense. Notice that the concept of DIMP is not applicable to a single cell (if we use an instrumentation/protocol to directly measure methylation at the molecular level, and not via PCR), since a concrete, single DNA cytosine position in a single cell is methylated or not methylated.

However, when pooling DNA extracted from a tissue, the previous reasonings about uncertainty hold plus an additional uncertainty factor: cells from the same tissue are not synchronized but are

found in different stages of their ontogenetic developments. Hence, the DIMP concept holds in the mentioned circumstances where the uncertainty of methylation is present.

Value

A list of GRanges containing only differentially informative position (DIMPs).

Examples

```
num.points <- 1000
HD <- GRangesList(
    sample1 = makeGRangesFromDataFrame(
        data.frame(chr = "chr1", start = 1:num.points, end = 1:num.points,
                strand = '*',
                hdiv = rweibull(1:num.points, shape = 0.75, scale = 1)),
        keep.extra.columns = TRUE),
    sample2 = makeGRangesFromDataFrame(
        data.frame(chr = "chr1", start = 1:num.points, end = 1:num.points,
                strand = '*',
                hdiv = rweibull(1:num.points, shape = 0.75, scale = 1)),
        keep.extra.columns = TRUE))

nlms <- nonlinearFitDist(HD, column = 1, verbose = FALSE)

PS <- getPotentialDIMP(LR = HD, nlms = nlms, div.col = 1, alpha = 0.05)
cutpoints <- estimateCutPoint(PS, control.names = "sample1",
                              treatment.names = c("sample2"),
                              div.col = 1, verbose = FALSE)
DIMPs <- selectDIMP(PS, div.col = 1, cutpoint = cutpoints$cutpoint$sample1)
```

---

sortBySeqnameAndStart *Sorting 'GRanges' objects*

Description

Sorts a GRanges object by seqname and start position

Usage

```
sortBySeqnameAndStart(gr)
```

Arguments

gr	GRanges object

Value

GRanges object

Examples

```
GR <- as(c("chr2:1-1", "chr1:1-1"), "GRanges")
GR <- sortBySeqnameAndStart(GR)
```

---

uniqueGRanges *Unique genomic ranges from a list of GRanges objects*

Description

Build an unique GRanges object from a list of Granges objects.

Usage

```
uniqueGRanges(ListOfGranges, ncols = NULL, columns = NULL,
  chromosomes = NULL, maxgap = -1L, minoverlap = 1L, missing = 0,
  type = c("any", "start", "end", "within", "equal"), select = c("all",
  "first", "last", "arbitrary"), ignore.strand = FALSE, num.cores = 1,
  tasks = 0L, verbose = TRUE)
```

Arguments

ListOfGranges	Objects to combine. A list of GRanges object or a GRangesList object.
ncols	integer. Number of columns to use from the meta-column of each GRanges object. Default value: NULL. If NULL, all the columns (from column 1 to ncols) from each GRanges will be present in the uniqueGRanges output.
columns	interger number(s) corresponding to the specific column(s) to use from the meta-column of each GRanges. Default value: NULL. if provided, the metacolumn from the uniqueGRanges output will contain the specified columns.
chromosomes	Chromosomes used Default value: NULL
maxgap	See GenomicRanges::findOverlaps in the IRanges package for a description of these arguments Default value: -1L
minoverlap	See GenomicRanges::findOverlaps in the IRanges package for a description of these arguments Default value: 1L
missing	A numerical value (default 0) or NA to write in ranges with missing values. For example, suppose that we want to build a uniqueGRanges object from the GRanges objects X and Y. If a given range k from the GRanges object X with metacolum value x is missing in the GRanges object Y, then the metacolum of range k from uniqueGRanges(list(X,Y)) object will be the row vector (x,0) or (x,NA) if missing = NA.
type	By default, any overlap is accepted. By specifying the type parameter, one can select for specific types of overlap. The types correspond to operations in Allen's Interval Algebra (see references). If type is start or end, the intervals are required to have matching starts or ends, respectively. While this operation seems trivial, the naive implementation using outer would be much less efficient. Specifying

	equal as the type returns the intersection of the start and end matches. If type is within, the query interval must be wholly contained within the subject interval. Note that all matches must additionally satisfy the minoverlap constraint described above. The maxgap parameter has special meaning with the special overlap types. For start, end, and equal, it specifies the maximum difference in the starts, ends or both, respectively. For within, it is the maximum amount by which the subject may be wider than the query.
`select`	When select is "all" (the default), the results are returned as a Hits object. Otherwise the returned value is an integer vector parallel to query (i.e. same length) containing the first, last, or arbitrary overlapping interval in subject, with NA indicating intervals that did not overlap any intervals in subject.
`ignore.strand`	When set to TRUE, the strand information is ignored in the overlap calculations. Default value: TRUE
`num.cores`	The number of cores to use, i.e. at most how many child processes will be run simultaneously (see bplapply function from BiocParallel package).
`tasks`	integer(1). The number of tasks per job. value must be a scalar integer >= 0L. In this documentation a job is defined as a single call to a function, such as bplapply, bpmapply etc. A task is the division of the X argument into chunks. When tasks == 0 (default), X is divided as evenly as possible over the number of workers (see MulticoreParam from BiocParallel package).
`verbose`	if TRUE, prints the function log to stdout

Details

The metadata of each one of these GRanges must have one or more columns to yield a unique GRanges object with metadata columns from the original GRanges objects. Otherwise, a unique GRanges object will be created without metadata columns. Additionally, all metadata must be the same class, e.g. all numeric or all characters, or all factor

Value

a GRanges object

Examples

```
dfChr1 <- data.frame(chr = "chr1", start = 11:15, end = 11:15,
                     strand = c("+","-","+","*","."), score = 1:5)
dfChr2 <- data.frame(chr = "chr1", start = 11:15, end = 11:15,
                     strand = c("+","-","+","*","."), score = 1:5)
dfChr3 <- data.frame(chr = "chr1", start = 11:15, end = 11:15,
                     strand = c("+","-","+","*","."), score = 1:5)

gr1 <- makeGRangesFromDataFrame(dfChr1, keep.extra.columns = TRUE)
gr2 <- makeGRangesFromDataFrame(dfChr2, keep.extra.columns = TRUE)
gr3 <- makeGRangesFromDataFrame(dfChr3, keep.extra.columns = TRUE)

grList <- GRangesList("gr1" = gr1, "gr2" = gr2, "gr3" = gr3)

uniqueGRanges(grList)
```

uniqueGRfilterByCov *Unique GRanges of methylation read counts filtered by coverages*

Description

Given two GRanges objects, this function will filter by coverage each cytosine site from each GRanges object.

Usage

```
uniqueGRfilterByCov(x, y, min.coverage = 4, percentile = 0.9999,
  high.coverage = NULL, columns = c(mC = 1, uC = 2), num.cores = 1L,
  tasks = 0L, verbose = TRUE, ...)
```

Arguments

x	A GRanges object with methylated and unmethylated counts in its meta-column.
y	A GRanges object with methylated and unmethylated counts in its meta-column.
min.coverage	Cytosine sites with coverage less than min.coverage are discarded.
percentile	Threshold to remove the outliers from each file and all files stacked.
high.coverage	An integer for read counts. Cytosine sites having higher coverage than this are discarded.
columns	Vector of integer numbers of the columns (from each GRanges meta-column) where the methylated and unmethylated counts are provided. If not provided, then the methylated and unmethylated counts are assumed to be at columns 1 and 2, respectively.
num.cores	The number of cores to use, i.e. at most how many child processes will be run simultaneously (see bplapply function from BiocParallel package).
tasks	integer(1). The number of tasks per job. value must be a scalar integer >= 0L. In this documentation a job is defined as a single call to a function, such as bplapply, bpmapply etc. A task is the division of the X argument into chunks. When tasks == 0 (default), X is divided as evenly as possible over the number of workers (see MulticoreParam from BiocParallel package).
verbose	if TRUE, prints the function log to stdout
...	Additional parameters for 'uniqueGRanges' function.

Details

Cytosine sites with 'coverage' > 'min.coverage' and 'coverage' < 'percentile' (e.g., 99.9 percentile) in at least one of the samples are preserved. It is expected that the columns of methylated and unmethylated counts are given.

Value

A GRanges object with the columns of methylated and unmethylated counts filtered for each cytosine position.

Examples

```
dfChr1 <- data.frame(chr = "chr1", start = 11:15, end = 11:15,
strand <- c("+","-","+","*","."), mC = 1:5, uC = 1:5)
dfChr2 <- data.frame(chr = "chr1", start = 12:18, end = 12:18,
strand <- '*', mC = 1:7, uC = 1:7)
gr1 <- makeGRangesFromDataFrame(dfChr1, keep.extra.columns = TRUE)
gr2 <- makeGRangesFromDataFrame(dfChr2, keep.extra.columns = TRUE)
r1 <- uniqueGRfilterByCov(gr1, gr2, ignore.strand = TRUE)
```

---

unlist *Flatten lists extended to any class*

Description

Given a list 'x' of R objects from the same class and same format, unlist simplifies it to produce a new R object which contains all the initial components which in 'x' object.

Usage

```
unlist(x, recursive = TRUE, use.names = TRUE)
```

Arguments

x	Any list R object.

Details

This is a method to extend unlist generic function to handle any list of objects from the same class.

Examples

```
gr1 <-GRanges(seqnames = "chr2", ranges = IRanges(3, 6),
              strand = "+", score = 5L, GC = 0.45)
gr2 <-
  GRanges(seqnames = c("chr1", "chr1"),
          ranges = IRanges(c(7,13), width = 3),
          strand = c("+", "-"), score = 3:4, GC = c(0.3, 0.5))
gr3 <-
  GRanges(seqnames = c("chr1", "chr2"),
          ranges = IRanges(c(1, 4), c(3, 9)),
          strand = c("-", "-"), score = c(6L, 2L), GC = c(0.4, 0.1))
grl <- list("gr1" = gr1, "gr2" = gr2, "gr3" = gr3)
base::unlist(grl) # The default unlist does not work
unlist(grl)
```

validateClass *Function to validate S3 classes in MethylIT*

Description

Function to validate S3 classes in MethylIT

Usage

```
validateClass(LR, ...)

## Default S3 method:
validateClass(LR, ...)

## S3 method for class 'pDMP'
validateClass(LR)

## S3 method for class 'InfDiv'
validateClass(LR)
```

Arguments

LR	An object from class 'pDMP' or 'InfDiv'

weibull3P *Nonlinear fit of Weibull CDF*

Description

This function performs the nonlinear fit of Weibull CDF of a variable x

Usage

```
weibull3P(X, sample.size = 20, npoints = NULL, npoints0 = NULL,
  maxiter = 1024, tol = 1e-12, ftol = 1e-12, ptol = 1e-12,
  minFactor = 10^-6, verbose = TRUE, ...)
```

Arguments

X	numerical vector
sample.size	size of the sample
npoints	number of points used in the fit
npoints0	subset of points where to estimate the estimateECDF (used only to reduce computational time)

maxiter	positive integer. Termination occurs when the number of iterations reaches maxiter. Default value: 1024
tol	A positive numeric value specifying the tolerance level for the relative offset convergence criterion. Default value: 1e-12,
ftol	non-negative numeric. Termination occurs when both the actual and predicted relative reductions in the sum of squares are at most ftol. Therefore, ftol measures the relative error desired in the sum of squares. Default value: 1e-12,
ptol	non-negative numeric. Termination occurs when the relative error between two consecutive iterates is at most ptol. Therefore, ptol measures the relative error desired in the approximate solution. Default value: 1e-12,
minFactor	A positive numeric value specifying the minimum step-size factor allowed on any step in the iteration. The increment is calculated with a Gauss-Newton algorithm and successively halved until the residual sum of squares has been decreased or until the step-size factor has been reduced below this limit. Default value: 10^-6
verbose	if TRUE, prints the function log to stdout
...	other parameters

Details

The script algorithm first try to fit the two-parameter Weibull CDF (Weibull2P). If Weibull2P did not fit, then the algorithm will try to fit Weibull3P. The Levenberg-Marquardt algorithm implemented in 'minpack.lm' R package is used to perform the nonlinear fit. Cross-validations for the nonlinear regressions (R.Cross.val) were performed in each methylome as described in reference [1]. In addition, Stein's formula for adjusted R squared (rho) was used as an estimator of the average cross-validation predictive power [1].

Value

Model table with coefficients and goodness-of-fit results: Adj.R.Square, deviance, AIC, R.Cross.val, and rho, as well as, the coefficient covarianza matrix.

Author(s)

Robersy Sanchez - 06/03/2016

References

1. Stevens JP. Applied Multivariate Statistics for the Social Sciences. Fifth Edit. Routledge Academic; 2009.

Examples

```
x <- rweibull(1000, shape=0.75, scale=1)
weibull3P(x, sample.size = 100)
```

Index