Package 'MethylIT'

July 5, 2018

Title Methylation Analysis Based on Information Thermodynamics

Version 0.3.1

Encoding UTF-8

Author Robersy Sanchez

Maintainer Thomas Maher <twm118@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 6.0.1

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)
```

`bootstrap2x2` *bootstrap2x2*

Description

Parametric Bootstrap of 2x2 Contingence independence test. The goodness of fit statistic is the root-mean-square statistic (RMST) or Hellinger divergence, as proposed by Perkins et al. [1, 2]. Hellinger divergence (HD) is computed as proposed in [3].

Usage

```
bootstrap2x2(x, stat = "rmst", num.permut = 100)
```

Arguments

`x`	A numerical matrix corresponding to cross tabulation (2x2) table (contingency table).
`stat`	Statistic to be used in the testing: ’RMST’ or ’HD’
`num.permut`	Number of permutations.

Value

A p-value probability

References

1. Perkins W, Tygert M, Ward R. and Classical Exact Tests Often Wildly Misreport Significance; the Remedy Lies in Computers [Internet]. Uploaded to ArXiv. 2011. Report No.: arXiv:1108.4126v2. 2. Perkins, W., Tygert, M. & Ward, R. Computing the confidence levels for a root-mean square test of goodness-of-fit. 217, 9072-9084 (2011). 3. Basu, A., Mandal, A. & Pardo, L. Hypothesis testing for two discrete populations based on the Hellinger distance. Stat. Probab. Lett. 80, 206-214 (2010).

Examples

```
set.seed(123)
TeaTasting = matrix(c(8, 350, 2, 20), nrow = 2,
                    dimnames = list(Guess = c("Milk", "Tea"),
                    Truth = c("Milk", "Tea")))
## Small num.permut for test's speed sake
bootstrap2x2( TeaTasting, stat = "all", num.permut = 100 )
```

`cluster` *Clustering fuction to highlight cluster*

Usage

```
cluster(GR = NULL, matrix = NULL, dist.matrix = NULL,
  distance = "euclidean", agglo.method = NULL, transpose = FALSE,
  absolute = FALSE, labels = FALSE, dendo.params = FALSE, dendo.cex = 1,
  dendo.color = c(rgb(1, 0, 0, 0.1), rgb(0, 0, 1, 0.1)), num.cuts.dendo = 2,
  dendo.border = c(rgb(1, 0, 0, 0.1), rgb(0, 0, 1, 0.1)), dendo.mar = c(4,
  4, 0.1, 0), dendo.font = 1, dendo.cuts = c(0.66, 0, 0.33, 0),
  dendo.lwd = 0.25, ...)
```

Arguments

`matrix`	= NULL
`dist.matrix`	= NULL
`distance`	= "euclidean"
`agglo.method`	= NULL
`transpose`	= FALSE
`absolute`	= FALSE
`labels`	= FALSE
`hcluster`	= FALSE

`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,
  NormalizeContsFiltr = FALSE, CountPerBp = NULL, minCountPerIndv = 3,
  FilterLog2FC = TRUE, pAdjustMethod = "BH", pvalCutOff = 0.05,
  MVrate = 0.98, Minlog2FC = 0.5, 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"
`NormalizeContsFiltr`	if TRUE, the counts are normalized
`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
`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
`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.

Examples

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

`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, div.col = NULL,
  verbose = TRUE)
```

Arguments

`LR`	A list of GRanges objects containing a divergence variable used to perform ROC analysis and estimate the cutpoint
`control.names`	Names/IDs of the control samples. Each GRanges object must correspond to a sample, for example, sample 's1'. Then this sample can be accessed in the list of GRanges objects as LR$s1.
`treatment.names`	Same type and function as 'control.names'.
`div.col`	Column number for divergence variable used in the ROC analysis and estimation of the cutpoints.
`verbose`	If TRUE, prints the function log to stdout.

Details

The function performs: 1) A logistic regression using the potential signals from control and treatment to yield a posterior classification of the signal. 2) Estimation of the optimal cutpoint from the area under the curve (AUC) of a receiver operating characteristic (ROC) built using the prior and posterior classifications of the signal. 3) Selection of the cytosine positions with values of the given information divergence greater than the cutpoint.

In this context, the AUC is the probability of being able to distinguish the biological regulatory signal naturally generated in the control from that one induced by the treatment. The cytosine sites carrying a methylation signal shall be called differentially informative methylated positions (DIMP). Now, the probability that a DIMP is not induced by the treatment is given by the probability of false alarm (PFA, false positive). That is, the biological signal is naturally present in the control as well as in the treatment.

Value

A list of three matrices cutpoint matrix values, AUC matrix values, and accuracy matrices values. These matrices values derives from all possible ROC analysis: control sample_i versus treatment sample_j (i,j = 1,2, ...).

Examples

```
set.seed(123)
num.points <- 1000

## A list of GRanges objects with simulated Hellinger divergences in
## their metacolumns.
HD <- GRangesList(
    sample1 = makeGRangesFromDataFrame(
                data.frame(chr = "chr1", start = 1:num.points,
                        end = 1:num.points, strand = '*',
                        hdiv = rweibull(1:num.points, shape = 0.45,
                                scale = 0.2)),
                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))

## Nonlinear fit of Weiblul distribution
nlms <- nonlinearFitDist(HD, column = 1, verbose = FALSE)

## Estimation of the potential signal and cutpoints
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)
```

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

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 Bayessian = TRUE, the results are based on the posterior estimations of methylation levels.

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), 'evaluateDIMP-class' will return the evaluation of the classification of DIMPs into two clases: DIMPS from control and DIMPs from treatment samples. Notice that both groups of DIMPs are methylation regulatory signals. That is, these methylation changes do not follow the Weibull distribution deduced in [1] on statistical biophysical basis.

Usage

```
evaluateDIMPclass(LR, control.names, treatment.names, column = c(hdiv = FALSE,
  TV = FALSE, wprob = FALSE, pos = FALSE), classifier = c("logistic",
  "pca.logistic", "lda", "svm", "qda", "pca.lda", "pca.qda"), n.pc = 1,
  center = FALSE, scale = FALSE, interaction = NULL,
  output = "conf.mat", prop = 0.6, num.boot = 100, mc.cores = 1L,
  tasks = 0L, cachesize = 250007, tolerance = 1e-04,
  svm.kernel = c("linear", "polynomial", "radial", "sigmoid"), 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. 'SVM' applies Support Vector Machines classifier from R package e1071.
`n.pc`	Number of principal components (PCs) to use in the LDA. Only used if classifier = "pcaLDA". 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).
`mc.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).
`cachesize`	To be use in SVM. Cache memory in MB (default 40).
`tolerance`	Only used for SVM classifeier. tolerance of termination criterion (default: 0.001)
`svm.kernel`	The kernel used in training and predicting in SVM classifier. You might consider changing some of the following parameters, depending on the kernel type: "linear", "polynomial", "radial", "sigmoid" (see ?svm).
`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 uncorreleted 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 split 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 randomly 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 = FALSE,
                            wprob = FALSE, 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)
```

`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 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

```
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

Usage

```
getDIMPatGenes(GR, GENES, ignore.strand = TRUE)
```

Arguments

`GR`	A GRanges object with the coordinates of DIMPs
`GENES`	A GRanges object with gene coordinates and gene IDs. A meta-column named 'gene_id' carying the gene ids must be included.
`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 = '*',
                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 = 2.1)),
        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
```

`getGRegionsStat-methods`
Statistic of Genomic Regions

Description

A function to estimate the centrality measures of a specified variable given in GRanges object (a column from the metacolums of the GRanges object) after split the GRanges object into intervals.

Usage

```
getGRegionsStat(GR, win.size = 350, step.size = 350, grfeatures = NULL,
  stat = c("sum", "mean", "gmaean", "median", "density"), absolute = FALSE,
  select.strand = NULL, column = 1L, prob = FALSE, entropy = FALSE,
  maxgap = -1L, minoverlap = 0L, scaling = 1L, type = c("any", "start",
  "end", "within", "equal"), ignore.strand = FALSE, na.rm = TRUE)

## S4 method for signature 'GRanges'
getGRegionsStat(GR, win.size = 350, step.size = 350,
  grfeatures = NULL, stat = c("sum", "mean", "gmaean", "median", "density"),
  absolute = FALSE, select.strand = NULL, column = 1L, prob = FALSE,
  entropy = FALSE, maxgap = -1L, minoverlap = 0L, scaling = 1L,
  type = c("any", "start", "end", "within", "equal"), ignore.strand = FALSE,
  na.rm = TRUE)

## S4 method for signature 'list'
getGRegionsStat(GR, win.size = 350, step.size = 350,
  grfeatures = NULL, stat = c("sum", "mean", "gmaean", "median", "density"),
  absolute = FALSE, select.strand = NULL, column = 1L, prob = FALSE,
  entropy = FALSE, maxgap = -1L, minoverlap = 0L, scaling = 1L,
  type = c("any", "start", "end", "within", "equal"), ignore.strand = FALSE,
  na.rm = TRUE)

## S4 method for signature 'GRangesList'
getGRegionsStat(GR, win.size = 350, step.size = 350,
  grfeatures = NULL, stat = c("sum", "mean", "gmaean", "median", "density"),
  absolute = FALSE, select.strand = NULL, column = 1L, prob = FALSE,
  entropy = FALSE, maxgap = -1L, minoverlap = 0L, scaling = 1L,
  type = c("any", "start", "end", "within", "equal"), ignore.strand = FALSE,
  na.rm = TRUE)
```

Arguments

`GR`	A Grange object with the variable of interest in its metacolumn.
`win.size`	An integer for the size of the windows/regions size of the intervals of genomics regions.
`step.size`	Interval at which the regions/windows must be defined
`grfeatures`	A GRanges object corresponding to an annotated genomic feature. For example, gene region, transposable elements, exons, intergenic region, etc. If provided, then parameters 'win.size' and step.size are ignored and the statistics are estimated for 'grfeatures'.
`stat`	Statistic used to estimate the summarized value of the variable of interest in each interval/window. Posible options are: "mean", geometric mean ("gmean"), "median", "density", and "sum" (default). Here, we define "density" as the sum of values from the variable of interest in the given region devided by the length of the region.
`absolute`	Optional. Logic (default: FALSE). Whether to use the absolute values of the variable provided
`select.strand`	Optional. If provided,"+" or "-", then the summarized statistic is computed only for the specified DNA chain.

`column`	Integer number denoting the column where the variable of interest is located in the metacolumn of the GRanges object or an integer vector of two elements (only if prob = TRUE).
`prob`	Logic. If TRUE and the variable of interest has values between zero and 1, then the summarized statistic is comuputed using Fisher's transformation. If length(column) == 2, say with colums x1 and x2, then the variable of interest will be p = x1/(x1 + x2). For example, if x1 and x2 are methylated and unmethylated read counts, respectively, then p is the methylation level.
`entropy`	Logic. Whether to compute the entropy when prob == TRUE.
`maxgap, minoverlap, type`	See ?findOverlaps in the IRanges package for a description of these arguments.
`scaling`	integer (default 1). Scaling factor to be used when stat = "density". For example, if scaling = 1000, then density * scaling denotes the sum of values in 1000 bp.
`ignore.strand`	When set to TRUE, the strand information is ignored in the overlap calculations.
`na.rm`	Logical value. If TRUE, the NA values will be removed

Details

This function split a Grange object into intervals genomic regions (GR) of fixed size (as given in function "tileMethylCounts2" R package methylKit, with small changes). A summarized statistic (mean, median, geometric mean or sum) is calculated for the specified variable values from each region. Notice that if win.size == step.size, then non-overlapping windows are obtained.

Value

A GRanges object with the new genomic regions and their corresponding summarized statistic.

Author(s)

Robersy Sanchez

Examples

```
gr <- GRanges(seqnames = Rle( c("chr1", "chr2", "chr3", "chr4"),
            c(5, 5, 5, 5)),
            ranges = IRanges(start = 1:20, end = 1:20),
            strand = rep(c("+", "-"), 10),
            GC = seq(1, 0, length = 20))
grs <- getGRegionsStat(gr, win.size = 4, step.size = 4)
grs

## Selecting the positive strand
grs <- getGRegionsStat(gr, win.size = 4, step.size = 4, select.strand = "+")
grs

## Selecting the negative strand
grs <- getGRegionsStat(gr, win.size = 4, step.size = 4, select.strand = "-")
grs

## Operating over a list of GRanges objects
gr2 <- GRanges(seqnames = Rle( c("chr1", "chr2", "chr3", "chr4"),
                          c(5, 5, 5, 5)),
                ranges = IRanges(start = 1:20, end = 1:20),
                strand = rep(c("+", "-"), 10),
                GC = runif(20))

grs <- getGRegionsStat(list(gr1 = gr, gr2 = gr2), win.size = 4, step.size = 4)
```

`getMethContext` *Get Methylation Context from a chromosome DNA sequence*

Description

This function retrieves the methylation context from a chromosome DNA sequence in fasta format.

Usage

```
getMethContext(chr.seq, chromosome, verbose = TRUE)
```

Arguments

`chr.seq`	DNA sequence from a chromosome in fasta format.
`chromosome`	Chromosome name.
`verbose`	If TRUE, prints the function log to stdout

Value

GRanges object with three columns: 'trinucleotide', methylation context, and 'CHH' methylation subcontexts: 'CHA', 'CHC', and 'CHT'.

Examples

```
dna <- Biostrings::DNAString(x = "CCCTAACGACCCTAACGCTACCCTAAACCTCTGAAT",
    start = 1, nchar = NA)
getMethContext(chr.seq = dna, chromosome = "1", verbose = TRUE)
```

`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, div.col, alpha = 0.05, tv.col = NULL,
  tv.cut = NULL, min.coverage = NULL)
```

Arguments

`LR`	A list of GRanges objects. Each GRanges object carrying information divergence values for each cytosine site in its meta-column.
`nlms`	A list of Weibull distribution fitted models (output of 'fitNonlinearWeibullDist' function).
`div.col`	Column number for divergence variable is located in the meta-column.
`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

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 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

Cummulative density function (CDF) and random generation for the Generalized Gamma 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

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

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

Arguments

`n`	number of observations
`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).
`q`	numeric vector
`mu`	location parameter (numerical, default 0).
`lower.tail`	logical; if TRUE (default), probabilities are P[X<=x], otherwise, P[X > x]
`log.p`	logical; if TRUE, probabilities/densities p are returned as log(p).

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).

Value

probability gamma (3-parameters or 4-parameters) for pggamma or 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)
```

`heatmapChr` *Heatmap of GRanges Object*

Description

A function to create a Heatmap or a graphical representation of data where the individual values contained in a matrix are represented as colors of the GRanges object.

Usage

```
heatmapChr(GR, filename = NULL, chr, sample.id = NULL,
  factor.scale = 10^6, absolute = FALSE, xtitle = NULL, Barpalette,
  format = "tiff", width = 4000, height = 790, cex.scale = 1.5,
  fontfamily = "sans", mar.scale = c(2, 2, 2, 2), mgp.scale = c(3, 1, 0),
  mar.heatmap = c(2, 2, 2, 2), mgp.heatmap = c(3.5, 1, 0),
  compression = "lzw", res = 900, pointsize = 1, col.bar.lwd = 1,
  cex.heatmap = 1.5, cex.xaxis = 1.6, cex.yaxis = 2, cex.lab = 2,
  lwd.ticks = 0.5, cex.bar.lab = 2, xaxis.labels.pos = 0.1, oma = c(2,
  2, 2, 2), oma.scale = c(0, 0, 0, 0), xaxis.adj = c(0.5, 1),
  tick.breaks = 500, xline.label = NA, jpg.type = c("cairo", "cairo-png",
  "Xlib", "quartz"), dendo.params = NULL, cluster = NULL, ylas = 1,
  bar.las = 1, ...)
```

Arguments

`GR`	A Grange object with the variable of interest in its metacolumn.
`filename`	This is the name of the image file in which we want to output the Heatmap.
`chr`	This is a required argument which corresponds to the Chromosome of interest in the data.
`sample.id`	This is the id or column name which is the sample of the GRange object.
`factor.scale`	A number to scale chromosome position into "bp", "kbp", or "Mbp", depending on chromosome size
`absolute`	If absolute == TRUE, all the values taken for the Heatmap would be absolute values of the GRange object.
`xtitle`	This is the x axis title for the Heatmap which will be produced..
`Barpalette`	This is a required argument which defines the barpalette for the Heatmap which can be a colorRampPalette object.
`format`	This is the format of the output file which will have the Heatmap. Possible formats are "jpg", "png", "tiff" and "pdf". The default value for this is "tiff with compression = "lzw" and res = 600.
`width`	This is the width of the Heatmap image which will be produced.
`height`	This is the height of the Heatmap image which will be produced.
`cex.scale`	This is a number indicating the amount by which plotting text and symbols should be scaled relative to the default.

`fontfamily`	This value defines the name of the font family which will be used in text or labels for the Heatmap.
`mar.scale`	A numeric vector of length 4, which sets the margin sizes in the following order: bottom, left, top, and right.
`mgp.scale`	A numeric vector of length 3, which sets the axis label locations relative to the edge of the inner plot window. The first value represents the location the labels (i.e. xlab and ylab in plot), the second the tick-mark labels, and third the tick marks.
`compression`	This is the the type of compression to be used. The default compression type is "lzw".
`res`	This is the nominal resolution in ppi which will be recorded in the bitmap file, if a positive integer. Also used for units other than the default, and to convert points to pixels. The defualt resolution is 300ppi.
`pointsize`	The pointsize of plotted text, interpreted as big points (1/72 inch) at res ppi. The default value for this is 1.
`col.bar.lwd`	Line width grphical parameter for the color bar.
`cex.heatmap`	Cex for heatmap (not for the color bar).
`cex.xaxis`	Cex value for x-axis labels.
`cex.yaxis`	Cex value for y-axis labels.
`cex.lab`	Cex value for x-axis label size.
`lwd.ticks`	Line width for axis and ticks (heatmap only).
`cex.bar.lab`	Cex values for color bar labels.
`xaxis.labels.pos`	Y-coordinate for x-axis labels.
`oma, oma.scale`	Same as 'oma' graphical parameter (see ?par). 'oma' is used in the heatmap and oma.scale in the color bar.
`xaxis.adj`	Adjustment of the x-axis labels.
`tick.breaks`	An integer number used to introduce the number breaks in the chromosome scale where the tick will be located.
`xline.label`	specifying a value for xline.label overrides the default placement of x-axis title, and places them this many lines outwards from the plot edge
`jpg.type`	Paramter 'type' from 'jpeg' functions (see ?jpeg).
`cluster`	TO DO - MISSING
`ylas, bar.las`	numeric in 0,1,2,3; the style of y-axis and colo-bar labels, as given for graphical parameters "las" (?par).
`...`	Additional graphical parameters for 'par' R function used in the heatmap (not in the color bar).
`clust.plot`	TO DO - MISSING

Details

This function creates a Heatmap is a false color image with a color scale added to the right side and a chromosome scale to the bottom.

Value

A GRanges object with the new genomic regions and their corresponding summarized statistic.

Author(s)

Robersy Sanchez

Examples

```
set.seed(123)
## An auxiliary function to generate simulated hypothetical values from a
## variable with normal distribution

hypDT <- function(mean, sd, n, num.pos, noise = 20) {
    h <- hist(rnorm(n, mean = mean, sd = sd), breaks = num.pos, plot = FALSE)
    hyp <- h$density * 60 + runif(length(h$density)) * noise
    return(hyp)
}

mean <- 12
sd <- 2

## To add some noise
noise <- c(4, 10)
noise2 <- list(c(5, 5), c(6, 6))

## To generate a matrix of values with variations introduced by noise
hyp <- lapply(1:2, function(k) {
     h <- hypDT(mean = mean, sd = sd, n = 10^5,
                num.pos = 8000, noise = noise[k])
    h1 <- h + runif(length(h)) * noise2[[k]][1]
    h2 <- h + runif(length(h)) * noise2[[k]][2]
    h <- h + runif(length(h)) * noise2[[k]][1]
    return(cbind(h, h1, h2))
})

## A GRanges object is built, which will carries the previous matrix on its
## meta-columns
min.length <- min(unlist(lapply(hyp, nrow)))
hyp <- lapply(hyp, function(h) h[1:min.length,])
hyp <- do.call(cbind, hyp)
starts <- seq(0, 30000, 3)[1:min.length]
ends <- starts + 2
GR <- GRanges(seqnames = "chr1", ranges = IRanges(start = starts,
                end = ends))
mcols(GR) <- data.frame(hyp = hyp)
colnames(mcols(GR)) <- c("CT1", "CT2", "CT3", "TT1", "TT2", "TT3")

## Pallette used in the bar color
bar.palette <- colorRampPalette(c(rep("cyan",4), "green",rep("yellow", 2),
                                  rep("red", 3), rep("darkblue", 2),
                                  rep("black",2)), bias = 1.1, space = "rgb")

## Heatmap construction
file.remove(paste0(file, ".tiff"))
```

`jensenSDiv` *Compute Jensen-Shannon Divergence*

Description

Compute Jensen-Shannon Divergence of probqbility vectors p and q.

Usage

```
jensenSDiv(p, q, Pi = 0.5, logbase = 2)
```

Arguments

`p, q`	Probability vectors, sum(p_i) = 1 and sum(q_i) = 1.
`Pi`	Weight of the probability distribution p. The weight for q is: 1 - Pi. Default Pi = 0.5.
`logbase`	A positive number: the base with respect to which logarithms

Details

The Jensen–Shannon divergence is a method of measuring the similarity between two probability distributions. Here, the generalization given in reference [1] is used. Jensen–Shannon divergence is expressed in terms of Shannon entroppy. 0 < jensenSDiv(p, q) < 1, provided that the base 2 logarithm is used in the estimation of the Shannon entropies involved.

References

1. J. Lin, “Divergence Measures Based on the Shannon Entropy,” IEEE Trans. Inform. Theory, vol. 37, no. 1, pp. 145–151, 1991.

Examples

```
set.seed(123)
counts = sample.int(10)
prob.p = counts/sum(counts)
counts = sample.int(12,10)
prob.q = counts/sum(counts)
jensenSDiv(prob.p, prob.q)
```

`ksTest` *Kolmogorov-Smirnov statistics*

Description

Permutation test for Kolmogorov-Smirnov statistics

Usage

```
ksTest(x, CDF = "Weibull", pars, num.sampl = 999, sample.size,
  numcores = 1, verbose = TRUE, ...)
```

Arguments

`x`	numerical vector to perform the goodness of fit
`CDF`	the name of the cummulative distribution function (CDF)
`pars`	vector of parameters to evaluate the CDF: 4P GG distribution: c(shape=value, scale=value, mu=value, psi=value) 3P GG distribution: c(shape=value, scale=value, psi=value) 3P Weibull distribution: c(shape=value, scale=value, mu=value) 2P Weibull distribution: c(shape=value, scale=value)
`num.sampl`	number of elements to be sampled
`sample.size`	number of permutations. If sample.size < length(x), then the test becomes a Monte Carlo test
`numcores`	number of cores
`verbose`	If TRUE, prints the function log to stdout
`...`	other parameters

Value

gamma distribution CDF

Author(s)

Robersy Sanchez - 02/29/2016

References

Alastair Sanderson. Using R to analyse data statistical and numerical data analysis with R http://www.sr.bham.ac.uk/~ajrs
analyse_data.html

Examples

```
num.samples <- 1000
x <- rweibull(num.samples, shape = 1.01, scale = 1.01)
ksTest(x, pars = c(shape = 1, scale = 1))
```

`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, 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"), 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).
`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 script algorithm prepares the information divergence variable to try the fitting Weibull or generalized gamma distribution model to the data. If Weibull distribution is selected (default: "Weibull"), function 'Weibull3P' 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 dsitribution 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.

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)

predict.pcaLDA(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)

predict.pcaLogisticR(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)

predict.pcaQDA(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 'LogisticR' method*

Description

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

Usage

```
predict.LogisticR(object, newdata, type = c("class", "probabilities", "all"),
  ...)
```

Arguments

`object`	To use with function 'predict'. A 'glm' object from a logistic model 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

This function is specific for predictions based on a logistic model given by function 'evaluateDIMP-class'. A logistic model 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 probabilities.

`predictDIMPclass` *Predict DIMP class*

Description

This function classify each DIMP as a control or a treatment DIMP

Usage

```
predictDIMPclass(LR, model, conf.matrix = FALSE, control.names = NULL,
  treatment.names = NULL)
```

Arguments

`LR`	A list of GRanges objects obtained through the through MethylIT downstream analysis. Basically, this object is a list of GRanges containing only differentially informative position (DIMPs). The metacolumn of each GRanges must contain the columna: Hellinger divergence "hdiv", total variation "TV", the probability of potential DIMP "wprob", which naturally are added in the downstream analysis of MethylIT.
`model`	A classifier model obtained with the function 'evaluateDIMPclass'.
`conf.matrix`	Optional. Logic, whether a confusion matrix should be returned (default, FALSE, see below).
`control.names`	Optional. Names/IDs of the control samples, which must be include in thr variable LR (default, NULL).
`treatment.names`	Optional. Names/IDs of the treatment samples, which must be include in the variable LR (default, NULL).

Details

Predictions only makes sense if the query DIMPs belong to same methylation context and derive from an experiment accomplished under the same condition set for the DIMPs used to build the model.

Value

The same LR object with a column named "class" added to a GRanges object from LR (default). Based on the model prediction each DIMP is labeled as control "CT" or as treatment "TT". If "conf.matrix" is TRUE and the arguments control.names and treatment.names are provided, then the overall confusion matrix is returned

Examples

```
# Random generation of Hellinger divergence values from a Weibul
# distribution model and estimating their tail probabilities.
num.points <- 5000
set.seed(123)
hdiv11 = rweibull(1:num.points, shape = 0.45, scale = 1.2)
wprob11 = pweibull(hdiv11, shape = 0.45, scale = 1.2, lower.tail = FALSE)
hdiv12 = rweibull(1:num.points, shape = 0.45, scale = 1.2)
wprob12 = pweibull(hdiv12, shape = 0.45, scale = 1.2, lower.tail = FALSE)
hdiv21 = rweibull(1:num.points, shape = 0.6, scale = 1.02)
wprob21 = pweibull(hdiv21, shape = 0.6, scale = 1.02, lower.tail = FALSE)
hdiv22 = rweibull(1:num.points, shape = 0.61, scale = 1.02)
wprob22 = pweibull(hdiv22, shape = 0.61, scale = 1.02, lower.tail = FALSE)
#' Potential signal
PS <- GRangesList(
      sample11 = makeGRangesFromDataFrame(
          data.frame(chr = "chr1", start = 1:num.points, end = 1:num.points,
                     strand = '*', hdiv = hdiv11, wprob = wprob11),
          keep.extra.columns = TRUE),
          sample12 = makeGRangesFromDataFrame(
          data.frame(chr = "chr1", start = 1:num.points, end = 1:num.points,
                     strand = '*', hdiv = hdiv12, wprob = wprob12),
          keep.extra.columns = TRUE),
      sample21 = makeGRangesFromDataFrame(
          data.frame(chr = "chr1", start = 1:num.points, end = 1:num.points,
                     strand = '*', hdiv = hdiv21, wprob = wprob21),
          keep.extra.columns = TRUE),
          sample22 = makeGRangesFromDataFrame(
          data.frame(chr = "chr1", start = 1:num.points, end = 1:num.points,
                     strand = '*', hdiv = hdiv22, wprob = wprob22),
          keep.extra.columns = TRUE))
cutpoint = 5.76
DIMPs = selectDIMP(PS, div.col = 1, cutpoint = cutpoint)

#' A classification model can be fitted as follow:
conf.mat <- evaluateDIMPclass(DIMPs,
                              column = c(hdiv = TRUE, TV = FALSE,
                                         wprob = FALSE, pos = FALSE),
                              interaction = "wprob:hdiv",
                              control.names = c("sample11", "sample12"),
                              treatment.names = c("sample21", "sample22"))
# Now predictions of DIMP for control and treament can be obtained
pred = predictDIMPclass(LR = DIMPs, model = conf.mat$model,
                              conf.matrix = TRUE,
                              control.names = c("sample11", "sample12"),
                              treatment.names = c("sample21", "sample22"))
pred
```

`propTest` *Beta Regression for methylation levels and rates*

Usage

```
propTest(GR, control.names, treatment.names, link = "logit", type = "ML",
  num.cores = 1, tasks = 0L, verbose = TRUE)
```

Arguments

`GR`	GRanges objects including control and treatment samples containing the methylation levels. The name for each column must coincide with the names given for parameters: 'control.names' and 'treatment.names'.
`control.names`	Names/IDs of the control samples, which must be include in the variable GR at the metacolumn.
`treatment.names`	Names/IDs of the treatment samples, which must be included in the variable GR at the metacolumn.
`link`	Parameter to be passed to function 'betareg' from package 'betareg'. character specification of the link function in the mean model (mu). Currently, "logit", "probit", "cloglog", "cauchit", "log", "loglog" are supported. Alternatively, an object of class "link-glm" can be supplied.
`type`	Parameter to be passed to function 'betareg' from package 'betareg'. A character specification of the type of estimator. Currently, maximum likelihood ("ML"), ML with bias correction ("BC"), and ML with bias reduction ("BR") are supported.
`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
`mc.cores`	The number of cores to use, i.e. at most how many child processes will be run simultaneously (see bpapply function from BiocParallel).

Details

Beta Regression analysis for group comparison of methylation levels is performed using the function 'betareg' from package 'betareg'.

Value

The original GRanges object with the columns "beta", "log2FC", and "pvalue" added.

Examples

```
num.cyt <- 11001 # Number of cytosine position with methylation call
max.cyt = 14000
## 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 = 3000:max.cyt,
             end = 3000:max.cyt,
             strand = '*',
             p1 = rbeta(num.cyt, 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 = 3000:max.cyt,
               end = 3000:max.cyt,
               strand = '*',
               p2 = rbeta(num.cyt, shape1 = 1.5, shape2 = 2)),
    keep.extra.columns = TRUE),
  sample12 = makeGRangesFromDataFrame(
    data.frame(chr = '1',
               start = 3000:max.cyt,
               end = 3000:max.cyt,
               strand = '*',
               p2 = rbeta(num.cyt, shape1 = 1.6, shape2 = 2.1)),
    keep.extra.columns = TRUE),
  sample21 = makeGRangesFromDataFrame(
    data.frame(chr = '1',
               start = 3000:max.cyt,
               end = 3000:max.cyt,
               strand = '*',
               p2 = rbeta(num.cyt, shape1 = 10, shape2 = 4)),
    keep.extra.columns = TRUE),
  sample22 = makeGRangesFromDataFrame(
    data.frame(chr = '1',
               start = 3000:max.cyt,
               end = 3000:max.cyt,
               strand = '*',
               p2 = rbeta(num.cyt, 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 statistics genes
p_DIMPs = getGRegionsStat(GR = DIMPs, grfeatures = genes, stat = "mean",
                          prob = TRUE, column = 2L)

GR_p_DIMP = uniqueGRanges(p_DIMPs, type = "equal", chromosomes = "1")
colnames(mcols(GR_p_DIMP)) <-  c("sample11", "sample12", "sample21",
                                "sample22")
names(GR_p_DIMP) <- DIMR$GeneID

## Group differences between methylation levels
propTest(GR = GR_p_DIMP, control.names = c("sample11", "sample12"),
         treatment.names = c("sample21", "sample22"))
```

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, cutpoint)
```

Arguments

LR	A list of GRanges objects including control and treatment GRanges. Each GRanges object must correspond to a samples. For example, if a samples 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.
cutpoint	Cutpoint to select DIMPs. Cytosine positions with divergence greater than 'cutpoint' will selected as DIMPs. Cutpoints are estimated with the function 'estimateCutPoint'.

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 basically 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 direct measuring methylation at molecular level, and no 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 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)
```

`shannonEntr` *Compute Shannon Entropy*

Description

Compute Shannon Entropy of probqbility vector p.

Usage

```
shannonEntr(p, logbase = 2)
```

Arguments

`p`	A probability vector, sum(p) = 1.
`logbase`	A positive number: the base with respect to which logarithms

Details

By definition, if p_i = 0 for some i, the value of the corresponding summand 0*log(0) is taken to be 0.

Examples

```
counts = sample.int(10)
prob = counts/sum(counts)
shannonEntr(prob)
```

`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
`ncols`	Number of columns Default value: NULL or all factor.
`columns`	interger number(s) corresponding to the specific column(s) to use from the metacolumn 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
`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.

Details

The metadata of each one of these GRanges must one or more columns. 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)
```

`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