Package 'MethylIT'

November 5, 2021

Title Methylation Analysis Based on Signal Detection

Version 0.3.2.3

Encoding UTF-8

URL https://github.com/genomaths/MethylIT

Description Methyl-IT implements methylation analysis based on signal detection theory and machine-learning. Methylation process is a stochastic process, particularly, a biochemical-biophysical process which must not be reduced to statistic. Methyl-IT includes the information on the statistical biophysics of the methylation process with the estimation of the probability distribution of the noise (plus signal), which is expressed in terms of information divergences.

Depends R (>= 3.6.0), rtracklayer, rmarkdown

License file LICENSE

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

LazyData yes

Imports BiocGenerics, BiocParallel, caret, data.table, genefilter, GenomeInfoDb, GenomicRanges, graphics, IRanges, matrixStats, MASS, methods, minpack.lm, nls2, RCurl, S4Vectors, stats, utils, XML

RoxygenNote 7.1.1

Suggests spelling, testthat, markdown, knitr, BiocStyle

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

Language en-US

BugReports https://github.com/genomaths/MethylIT/issues

NeedsCompilation no

Author Robersy Sanchez [aut, cre] (<https://orcid.org/0000-0002-5246-1453>),
Jose Raul Barreras [ctb],
Thomas Maher [ctb]

Maintainer Robersy Sanchez <rus547@psu.edu>

R topics documented:

`countTest2` *Regression Test for Count*

Description

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

Usage

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

Arguments

`DS`	A 'glmDataSet' object, which is created with function `glmDataSet`.

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

countFilter
: whether or not to filter the counts according to the minimum count per region per each individual/sample, which is setting by 'minCountPerIndv'.

CountPerBp
: for each group the count per bp must be equal or greater than CountPerBp. The filter is applied if 'CountPerBp' is given and if 'x' DESeqDataSet object has the rowRanges as a GRanges object on it

minCountPerIndv
: each gene or region must have more than 'minCountPerIndv' counts (on average) per individual in at least one group.

maxGrpCV
: A numerical vector. Maximum coefficient of variance for each group. Default maxGrpCV = NULL. The numbers 'maxGrpCV1' and 'maxGrpCV2' will be taken as the maximun variances values permitted in control and in treatment groups, respectively. If only 'maxGrpCV1' is provided, then maxGrpCV = c('maxGrpCV1', 'maxGrpCV1'). This parameter is addressed to prevent testing regions where intra-group variations are very large, e.g.: control = c(1,0,1,1) and treatment = c(1, 0, 1, 40). The coefficient of variance for the treatment group is 1.87, very high. The generalized linear regression analysis would yield statistical significant group differences, but evidently there is something wrong in one of the treatment samples. We would try the application of further statistical smoothing approach, but we prefer to leave the user decide which regions to test.

FilterLog2FC
: if TRUE, the results are filtered using the minimum absolute value of log2FoldChanges observed to accept that a gene in the treatment is differentially expressed in respect to the control

pAdjustMethod
: method used to adjust the results; default: BH

pvalCutOff
: cutoff used then a p-value adjustment is performed.

MVrate
: Minimum Mean/Variance rate.

Minlog2FC
: minimum logarithm base 2 of fold changes.

test
: A character string matching one of 'Wald' or 'LRT'. If test = 'Wald', then the p-value of the Wald test for the coefficient of the independent variable (*treatment group*) will be reported. If test = 'LRT', then the p-value from a likelihood ratio test given by anova function from *stats* packages will be the reported p-value for the group comparison when the best fitted model is the negative binomial. As suggested for glm, if best fitted model is Poisson or quasi-Poisson, then the best test is 'Chi-squared' or 'F-test', respectively. So, for the sake of simplicity, the corresponding suitable test will be applied when test = 'LRT'.

scaling
: integer (default 1). Scaling factor estimate the signal density as: scaling x 'DMP-Count-Per-Bp'. For example, if scaling = 1000, then signal density denotes the number of DMPs in 1000 bp.

saveAll
: if TRUE all the temporal results are returned

verbose
: if TRUE, prints the function log to stdout

Details

A pairwise group comparison, control versus treatment, is performed. The experimental design settings must be introduced using function `glmDataSet` to provide dataset (DS) object.

Value

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

See Also

`glmDataSet`

Examples

```
set.seed(133) # Set a seed
## A GRanges object with the count matrix in the metacolumns is created
countData <- matrix(sample.int(200, 500, replace = TRUE), ncol = 4)
colnames(countData) <- c('A1','A2','B1','B2')

start <- seq(1, 25e4, 2000)
end <- start + 1000
chr <- c(rep('chr1', 70), rep('chr2', 55))
GR <- GRanges(seqnames = chr, IRanges(start = start, end = end))
mcols(GR) <- countData

## Gene IDs
names(GR) <- paste0('gene', 1:length(GR))

## An experiment design is set.
colData <- data.frame(condition = factor(c('A','A','B','B')),
                    c('A1','A2','B1','B2'), row.names = 2)

## A RangedGlmDataSet is created
ds <- glmDataSet(GR = GR, colData = colData)

## The gneralized linear model pairwise group comparison, group 'A'
## ('control') versus 'B' (treatment) is performed.
countTest2(ds, num.cores = 1L, maxGrpCV = c(0.4, 0.4), verbose = FALSE)
```

---

cutpoint *Cutpoint of the 'PS' simulated dataset used in the examples*

Description

Each sample individual in the 'PS' included 10000 cytosine positions

Usage

```
cutpoint
```

Format

A list with the cutpoint value and classification performance. The cutpoint values was obtained with function `estimateCutPoint`.

---

dmpClusters *DMP clustering*

Description

Given a 'pDMP' (or 'InfDiv') object carrying DMPs (methylated cytosines) detected in Methyl-IT downstream analysis, function ***'dmpClusters'*** build clusters of DMPs, which can be further tested to identify differentially methylated regions (DMRs) with `countTest2` function.

Usage

```
dmpClusters(
  GR,
  maxDist = 3,
  minNumDMPs = 1,
  maxClustDist = NULL,
  method = c("relaxed", "fixed.int"),
  chromosomes = NULL,
  columns = 9L,
  ignore.strand = TRUE,
  num.cores = detectCores() - 1,
  tasks = 0L,
  verbose = TRUE,
  ...
)
```

Arguments

`GR`	An object from ***'pDMP'*** class, which is returned by `selectDIMP` function.
`maxDist`	maximum distance at which two reported bases sites from the same cluster can be separated. Default: $maxDist = 3$.
`minNumDMPs`	Minimum number of DMPs inside of each cluster. Default: $minNumDMPs = 1$.
`maxClustDist`	Clusters separated by a distance lesser than maxClustDist' positions are merged. Default: $maxClustDist = NULL$. If $0 < maxClustDist < maxDist$ or $maxClustDist = NULL$, then maxClustDist will be recalculated as $maxClustDist = maxDist + 1$.
`method`	Two different approaches are implemented to clustering DMPs:

"relaxed": DMP ranges which are separated by a distance less than *'maxClustDist'* are merged and ranges with less than *'minNumDMPs'* are removed.

"fixed.int": It will generate a partition where the distance between consecutive DMPs is not greater than *'maxDist'*. Ranges with less than *'minNumDMPs'* are removed. If, additionally, a value *maxClustDist > 0* is provided, then the "relaxed" approach is applied to the ranges from the first step.

`chromosomes` vector of characters labeling the chromosomes included in the analysis. Default: chromosomes = NULL (all chromosomes are included).

`columns` An integer number corresponding to the specific column(s) to use from the meta-column of each GRanges. Default values is 9 (the column carrying to the Hellinger divergence values).

`ignore.strand`
Same as in `findOverlaps-methods`.

`num.cores, tasks`
integer(1). The number of cores to use, i.e. at most how many child processes will be run simultaneously (see `bplapply` function from BiocParallel package).The number of tasks per job. value must be a scalar integer >= 0L (see MulticoreParam from BiocParallel package).

`verbose` if TRUE, prints the function log to stdout.

`...` Further parameters for `uniqueGRanges` function.

Details

Two algorithmic approaches are implemented, named: "relaxed" and "fixed.int" (see the description of parameter 'method'). The "fixed.int" is mostly addressed to find specific methylation patterns, but the price is the number of DMRs found is lower.

The number of DMPs reported in each cluster corresponds to the numbers of sites inside the cluster where DMPs were found in at least one of the samples (from control or from treatment). That is, **dmpClusters** is only a tool to locate regions with high density of DMPs from all the samples. It does not detect DMRs. It is assumed that only DMP coordinates are given in the *'GR'* object. That is, all the sites provided are considered in the computation.

Value

A GRanges object with the numbers of positions inside each cluster, where DMPs were reported in at least one of the samples.

Author(s)

Robersy Sanchez (https://genomaths.com).

References

1. Sanchez R, Mackenzie SA (2016) Information Thermodynamics of Cytosine DNA Methylation. PLOS ONE 11(3): e0150427. https://doi.org/10.1371/journal.pone.0150427

Examples

```
## Get a dataset of dmps from the package
data(dmps)

## Build clusters of DMPs taking into account the DNA strand
x1 = dmpClusters(GR = dmps, maxDist = 7, minNumDMPs = 6,
                 method = "fixed.int", ignore.strand = FALSE,
                 verbose = FALSE)
data.frame(x1)

## Not run:
## Build clusters of DMPs ignoring DNA strand and maxClustDist = 7
x2 = dmpClusters(GR = dmps, maxDist = 7, minNumDMPs = 6,
                  maxClustDist = 7, method = "fixed.int",
                  num.cores=2L, ignore.strand = TRUE,
                  verbose = FALSE)
DataFrame(data.frame(x2))

## The relaxed approach with method = "relaxed"
x3 = dmpClusters(GR = dmps, minNumDMPs = 6, method = "relaxed",
                  maxClustDist = 10, ignore.strand = TRUE,
                  verbose = FALSE)
DataFrame(data.frame(x3))

## ==== Setting up the experiment design to test for DMRs ===
nams <- names(dmps)
dmps_at_clusters <- getDMPatRegions(GR = dmps, regions = x3,
                                    ignore.strand = TRUE)
dmps_at_clusters <- uniqueGRanges(dmps_at_clusters, columns = 2L,
                                  ignore.strand = TRUE, type = 'equal',
                                  verbose = FALSE)
colnames(mcols(dmps_at_clusters)) <- nams

colData <- data.frame(condition = factor(c('CT', 'CT', 'CT',
                                           'TT', 'TT', 'TT'),
                                           levels = c('CT', 'TT')),
                      nams, row.names = 2)

## Build a RangedGlmDataSet object
ds <- glmDataSet(GR = dmps_at_clusters, colData = colData)

## ================ Testing to detect DMRs ===========
dmrs <- countTest2(DS = ds, num.cores = 4L, minCountPerIndv = 4,
                   maxGrpCV = c(1, 1), Minlog2FC = 0.5,
                   CountPerBp = 0.001, test = 'LRT',
                   verbose = TRUE)
dmrs

## End(Not run)
```

dmps *Simulated dataset of DMPs used in examples*

Description

Each individuals sample includes 10000 cytosine positions

Usage

```
dmps
```

Format

dmps is an object from class 'pDMP' carrying in the meta-columns the following variables:

p1 methylation level from the reference sample

p2 methylation level from the treatment sample

TV the total variation distance (difference of methylation levels)

hdiv Hellinger divergence

wprob the probabilities: $wprob = 1 - CDFprobability$

dmps is an object from class 'pDMP' carrying the same meta-columns as 'HD' (dataset) plus the probabilities: $wprob = 1 - CDFprobability$. ***DMPs*** ('dmps') are obtained from potential DMPs (see `PS`) with `selectDIMP` function.

ds *Simulated dataset of RangedGlmDataSet class object (DMPs counts)*

Description

RangedGlmDataSet and *glmDataSet* are objects carrying the information on the experimental design for two group comparison of DMP counts by applying generalized linear regression.

Usage

```
ds
```

Format

ds is an *RangedGlmDataSet* with 125 regions and 4 columns (individuals) with factor levels 'A' and 'B'. The accessible objects in the dataset are:

GR A `GRanges-class` object of length(GR) = 125 with the count matrix of DMPs in the meta-columns.

counts Count matrix of DMPs with minimal dim(ds$counts) = c(125, 4).

colData A data frame with one columnn named 'condition'.

sampleNames Samples names: "A1" "A2" "B1" "B2"

levels Design factor levels: "A" "B"

ds is an object from class *RangedGlmDataSet* carrying the information to perform DMG analysis with `countTest2` function.

`estimateBayesianDivergence`

Information divergence estimator

Description

The Information divergence of methylation levels is estimated using the direct estimation or a Bayesian approach of the methylation levels. if 'meth.level = FALSE', Hellinger divergence is computed as given in reference (1). The Bayesian approach followed is described in reference (2).

Usage

```
estimateBayesianDivergence(
  x,
  Bayesian = FALSE,
  init.pars = NULL,
  JD = FALSE,
  jd.stat = FALSE,
  num.cores = 1,
  tasks = 0L,
  columns = c(mC1 = 1, uC1 = 2, mC2 = 3, uC2 = 4),
  meth.level = FALSE,
  preserve.gr = FALSE,
  loss.fun = c("linear", "huber", "smooth", "cauchy", "arctg"),
  logbase = 2,
  verbose = TRUE,
  ...
)
```

Arguments

x	A matrix of counts or GRanges object with the table of counts in the meta-columns (methylated mC and unmethylated uC cytosines). Unless specified in the parameter 'columns', the methylation counts must be given in the first four columns: 'mC1' and 'uC1' methylated and unmethylated counts for control sample, and 'mC2' and 'uC2' methylated and unmethylated counts for treatment sample, respectively.
Bayesian	logical. Whether to perform the estimations based on posterior estimations of methylation levels.
init.pars	initial parameter values. Defaults is NULL and an initial guess is estimated using optim function. If the initial guessing fails initial parameter values are to alpha = 1 & beta = 1, which imply the parsimony pseudo-counts greater than zero.
JD	Logic (Default:FALSE). Option on whether to add a column with values of J-information divergence (see estimateJDiv). It is only compute if JD = TRUE and meth.level = FALSE.
jd.stat	logical(1). Whether to compute the JD statistic with asymptotic Chi-squared distribution with one degree of freedom (see estimateJDiv).
num.cores	The number of cores to use, i.e. at most how many child processes will be run simultaneously (see 'bplapply' function from BiocParallel package).
tasks	integer(1). The number of tasks per job. value must be a scalar integer >= 0L. In this documentation a job is defined as a single call to a function, such as bplapply, bpmapply etc. A task is the division of the X argument into chunks. When tasks == 0 (default), X is divided as evenly as possible over the number of workers (see MulticoreParam from BiocParallel package).
columns	Vector of integer numbers of the columns where the counts 'mC1', 'uC1', 'mC2', and 'uC2' are in the matrix (default 1 to 4). That is, the input could have more than 4 columns, but only 4 columns with the counts are used. if 'meth.level == TRUE', then columns = $c('p1', 'p2')$.
meth.level	logical(1). Methylation levels can be provided in place of counts.
preserve.gr	logical(1). Option of whether to preserve all the metadata from the original GRanges object.
loss.fun	Loss function(s) used in the estimation of the best fitted model to beta distribution (only applied when Bayesian=TRUE; see (Loss function)). This fitting uses the approach followed in in the R package usefr. After $z = 1/2 * sum((f(x) - y)^2)$ we have: 1. "linear": linear function which gives a standard least squares: $loss(z) = z$. 2. "huber": Huber loss, $loss(z) = ifelse(z \leq 1, z, sqrt(z) - 1)$. 3. "smooth": Smooth approximation to the sum of residues absolute values: $loss(z) = 2 * (sqrt(z + 1) - 1)$. 4. "cauchy": Cauchy loss: $loss(z) = log(z + 1)$. 5. "arctg": arc-tangent loss function: $loss(x) = atan(z)$. Loss 'linear' function works well for most of the methylation datasets with acceptable quality.

logbase	Logarithm base used to compute the JD (if JD = TRUE). Logarithm base 2 is used as default (bit unit). Use logbase = exp(1) for natural logarithm.
verbose	if TRUE, prints the function log to stdout
...	Optional parameter values for: maxiter, ftol, ptol, and gradtol from nlsLM and nlm functions.

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,

$$H = 2(n_1+1)(n_2+1)*((sqrt(p_1)-sqrt(p_2))^2+(sqrt(1-p_2)-sqrt(1-p_2))^2)/(n_1+n_2+2)$$

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:

$$H = (sqrt(p_1) - sqrt(p_2))^2 + (sqrt(1-p_1) - sqrt(1-p_2))^2$$

This formula assumes that the probability vectors derived from the methylation levels $p_i = c(p_{i2}, 1-p_{i2})$ (see function estimateHellingerDiv) are an unbiased estimation of the expected one. The Bayesian approach followed here is described in reference (2).

Value

The input matrix or GRanges object with the four columns of counts and additional columns. If Bayessian = TRUE, the results are based on the posterior estimations of methylation levels. The basic additional columns are:

1) The original matrix of methylated c_{ij} and unmethylated t_{ij} read counts from control $j = 1$ and treatment $j = 2$ samples at positions i.

2) 'p1' and 'p2': methylation levels for control and treatment, respectively. If 'meth.level = FALSE' and 'Bayesian = TRUE' (recommended), 'p1' and 'p2' are estimated following the Bayesian approach described in reference (1).

3) 'bay.TV': total variation TV = p2 - p1

4) 'TV': total variation based on simple counts: $TV = c_1/(c_1+t_1) - c_2/(c_2+t_2)$, where c_i and t_i denote methylated and unmethylated read counts, respectively.

5) 'hdiv': Hellinger divergence. If Bayesian = TRUE, the results are based on the posterior estimations of methylation levels. if 'meth.level = FALSE', then 'hdiv' is computed as given in reference (1), otherwise as:

$$hdiv = (sqrt(p_1) - sqrt(p_2))^2 + (sqrt(1-p_1) - sqrt(1-p_2))^2$$

Author(s)

Robersy Sanchez (https://genomaths.com)

References

1. Basu A., Mandal A., Pardo L. Hypothesis testing for two discrete populations based on the Hellinger distance. Stat. Probab. Lett. 2010, 80: 206-214.
2. Sanchez R, Yang X, Maher T, Mackenzie S. Discrimination of DNA Methylation Signal from Background Variation for Clinical Diagnostics. Int. J. Mol Sci, 2019, 20:5343.

See Also

estimateDivergence

Examples

```
## The read count data are created
gr <- data.frame(chr = 'chr1', start = 1:10, end = 1:10,
                 strand = '*',
                 mC1 = rnbinom(size = 10, mu = 4, n = 500),
                 uC1 = rnbinom(size = 10, mu = 4, n = 500),
                 mC2 = rnbinom(size = 10, mu = 4, n = 500),
                 uC2 = rnbinom(size = 10, mu = 4, n = 500))
gr <- makeGRangesFromDataFrame(gr, keep.extra.columns = TRUE)

## Estimation of the information divergences
hd <- estimateBayesianDivergence(gr, JD = TRUE)

## Keep in mind that Hellinger and J divergences are, in general,
## correlated
cor.test(x = as.numeric(hd$hdiv), y = as.numeric(hd$jdiv),
        method = 'kendall')

## An example with methylation levels
set.seed(123)
sites = 10
dat = data.frame(chr = 'chr1', start = 1:sites, end = 1:sites,strand = '*',
                 m1 = runif(n = sites), m2 = runif(n = sites))

dat =  makeGRangesFromDataFrame(dat, keep.extra.columns = TRUE)

## Transforming the list of data frames into a GRanges object
hd <- estimateBayesianDivergence(x = dat, columns = c(p1 = 1, p2 = 2),
                                 meth.level = TRUE, preserve.gr = TRUE)
```

`estimateBetaDist` *Select the beta distribution that fit specified quantiles*

Description

This function perform a nonlinear estimation of the shape parameters of beta distribution

Usage

```
estimateBetaDist(
  q,
  init.pars = NULL,
  loss.fun = c("linear", "huber", "smooth", "cauchy", "arctg"),
  maxiter = 1024,
  maxfev = 1e+05,
  ftol = 1e-12,
  ptol = 1e-12
)
```

Arguments

`q`	prior probabilities
`init.pars`	initial parameter values. Defaults is NULL and an initial guess is estimated using `nls.lm` function.
`loss.fun`	Loss function(s) used in the regression (see (Loss function)). This fitting uses the approach followed in in the R package usefr. After $z = 1/2 * sum((f(x) - y)^2)$ we have: 1. "linear": linear function which gives a standard least squares: $loss(z) = z$. 2. "huber": Huber loss, $loss(z) = ifelse(z \leq 1, z, sqrt(z) - 1)$. 3. "smooth": Smooth approximation to the sum of residues absolute values: $loss(z) = 2 * (sqrt(z + 1) - 1)$. 4. "cauchy": Cauchy loss: $loss(z) = log(z + 1)$. 5. "arctg": arc-tangent loss function: $loss(x) = atan(z)$.
`maxiter, ftol, ptol, maxfev`	Optional parameters for `nlsLM` and `nls.lm` functions.

Details

To obtain the estimates for shape parameters from the best fitted beta distribution model, a nonlinear regression Levenberg-Marquardt algorithm implemented in function `nls.lm` is applied. Several (loss functions)) are available to accomplish the model fitting to the data. If `nls.lm` function fails, then a new try will be accomplish with `nlsLM` function. If the previous algorithms fail, then the parameters will be estimated using BFGS" algorithm implemented in `optim` function.

Value

the estimated values of the shape parameters of the selected beta distribution.

Examples

```
## ------ A simple example -----
set.seed(4)
br <- rbeta(1e3, shape1 = 1, shape2 = 2)
pars <- estimateBetaDist(br)
pars
```

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

Description

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

Usage

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

Arguments

`LR`	An object from 'pDMP' class. This object is previously obtained with function `getPotentialDIMP`.
`control.names, treatment.names`	Names/IDs of the control and treatment samples, which must be include in the variable LR.
`simple`	Logic (default, TRUE). If TRUE, then Youden Index is used to estimate the cutpoint. If FALSE, the minimum information divergence value with posterior

classification probability greater than *post.cut* (usually *post.cut* = 0.5) as estimated by *classifier1* will be the reported cutpoint, except if a better cutpoint is found in the set of values provided by the user in the parameter *cut.values*.

`column` a logical vector for column names for the predictor variables to be used: Hellinger divergence 'hdiv', total variation 'TV', probability of potential DMP '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, respectively. If 'wprob = TRUE', then Logarithm base-10 of 'wprob' will be used as predictor in place of 'wprob'.

`classifier1, classifier2`
Classification model to use. Option 'logistic' applies a logistic regression model; option 'lda' applies a Linear Discriminant Analysis (LDA); 'qda' applies a Quadratic Discriminant Analysis (QDA), 'pca.logistic' applies logistic regression model using the Principal Component (PCs) estimated with Principal Component Analysis (PCA) as predictor variables. pca.lda' applies LDA using PCs as predictor variables, and the option 'pca.qda' applies a Quadratic Discriminant Analysis (QDA) using PCs as predictor variables. If classifier2 is not NULL, then it will be used to evaluate the classification performance, and the corresponding best fitted model will be returned.

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

`tv.col` Column number for the total variation to be used for filtering cytosine positions (if provided).

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

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

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

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

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

`interactions` If a logistic classifier is used. 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.

`cut.values` Cut values of the information divergence (ID) specified in *div.col* where to check the classification performance (0 < *cut.interval* < max ID). If provided, the search for a cutpoint will include these values.

`stat` An integer number indicating the statistic to be used in the testing when *simple* = FALSE The mapping for statistic names are:

- 0 = 'Accuracy'

- 1 = 'Sensitivity'
- 2 = 'Specificity'
- 3 = 'Pos Pred Value'
- 4 = 'Neg Pred Value'
- 5 = 'Precision'
- 6 = 'Recall'
- 7 = 'F1'
- 8 = 'Prevalence'
- 9 = 'Detection Rate'
- 10 = 'Detection Prevalence'
- 11 = 'Balanced Accuracy'
- 12 = FDR

`cutp_data` logical(1) (optional). If TRUE, and simple = TRUE, then a data frame for further analysis or estimation of the optimal cutpoint based only on the selected divergence is provided.

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

`...` arguments passed to or from other methods.

Details

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

Results of the classification performance resulting from the estimated cutpoint are normally given, with the exception of those extreme situations where the statistics to evaluate performance cannot be estimated. More than one classifier model can be applied. For example, one classifier (logistic model) can be used to estimate the posterior classification probabilities of DMP into those from control and those from treatment. These probabilities are then used to estimate the cutpoint in range of values from, say, 0.5 to 0.8. Next, a different classifier can be used to evaluate the classification performance. Different classifier models would yield different performances. Models are returned and can be used in further prediction with new datasets from the same batch experiment. This is a machine learning approach to discriminate the biological regulatory signal naturally generated in the control from that one induced by the treatment.

Notice that the estimation of an optimal cutpoint based on the application Youden Index (simple = TRUE) only uses the information provided by the selected information divergence. As a result, classification results based only in one variable can be poor or can fail. However, option simple = FALSE, uses the information from several variables following a machine-learning (ML) approach.

Nevertheless, when simple = TRUE, still a ML model classifier can be built using the optimal cutpoint estimated and setting clas.perf = TRUE. Such a ML model can be used for predictions in further analyses with function `predictDIMPclass`.

Value

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

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

See Also

evaluateDIMPclass

Examples

```
## Get a dataset of potential signals and the estimates cutpoint
## from the package and performs cutpoint estimation
data(PS)

cutp <- mlCutpoint(LR = PS,
                   column = c(hdiv = TRUE, TV = TRUE,
                              wprob = TRUE, pos = TRUE),
                   classifier1 = 'qda', n.pc = 4,
                   control.names = c('C1', 'C2', 'C3'),
                   treatment.names = c('T1', 'T2', 'T3'),
                   tv.cut = 0.68, prop = 0.6,
                   cut.values = seq(114, 118, 1),
                   div.col = 9L)
cutp
```

`estimateDivergence` *Information Divergences of Methylation Levels*

Description

This function prepares the data for the estimation of information divergences and works as a wrapper calling the functions that compute selected information divergences of methylation levels. In the downstream analysis, the probability distribution of a given information divergence is used in Methyl-IT as the null hypothesis of the noise distribution, which permits, in a further signal detection step, the discrimination of the methylation regulatory signal from the background noise.

For the current version, two information divergences of methylation levels are computed by default: 1) Hellinger divergence (*H*) and 2) the total variation distance (*TVD*). In the context of methylation analysis *TVD* corresponds to the absolute difference of methylation levels. Here, although the variable reported is the total variation (*TV*), the variable actually used for the downstream analysis is *TVD*. Once a differentially methylated position (DMP) is identified in the downstream analysis, *TV* is the standard indicator of whether the cytosine position is hyper- or hypo-methylated.

The option to compute the J-information divergence (JD) is currently provided. The motivation to introduce this divergence is given in the help of function `estimateJDiv`.

Usage

```
estimateDivergence(
  ref,
  indiv,
  Bayesian = FALSE,
  init.pars = NULL,
  columns = NULL,
  min.coverage = 4,
  min.meth = 4,
  min.umeth = 0,
  min.sitecov = 4,
  high.coverage = NULL,
  percentile = 0.999,
  JD = FALSE,
  jd.stat = FALSE,
  num.cores = detectCores(),
  tasks = 0L,
  meth.level = FALSE,
  loss.fun = c("linear", "huber", "smooth", "cauchy", "arctg"),
  logbase = 2,
  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(1). Whether to perform the estimations based on posterior estimations of methylation levels.

`init.pars` initial parameter values. Defaults is NULL and an initial guess is estimated using `optim` function. If the initial guessing fails initial parameter values are to alpha = 1 and beta = 1, which imply the parsimony pseudo-counts greater than zero.

`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` An integer or an integer vector of length 2. Cytosine sites where the coverage in both samples, 'x' and 'y', are less than 'min.coverage' are discarded. The cytosine site is preserved, however, if the coverage is greater than 'min.coverage' in at least one sample. If 'min.coverage' is an integer vector, then the corresponding min coverage is applied to each sample.

`min.meth` An integer or an integer vector of length 2. Cytosine sites where the numbers of read counts of methylated cytosine in both samples, '1' and '2', are less than 'min.meth' are discarded. If 'min.meth' is an integer vector, then the corresponding min number of reads is applied to each sample. Default is min.meth = 4.

`min.umeth` An integer or an integer vector of length 2 ($min.umeth = c(min.umeth1, min.umeth2)$). Min number of reads to consider cytosine position. Specifically cytosine positions where $(uC \leq min.umeth) and (mC > 0) and (mC \leq min.meth)$ hold will be removed, where mC and uC stand for the numbers of methylated and unmethylated reads. Default is $min.umeth = 0$.

`min.sitecov` An integer. The minimum total coverage. Only sites where the total coverage $(cov1 + cov2)$ is greater than 'min.sitecov' are considered for downstream analysis, where cov1 and cov2 are the coverages for samples 1 and 2, respectively.

`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.

`JD` Logic (Default:FALSE). Option on whether to add a column with values of J-information divergence (see `estimateJDiv`). It is only compute if JD = TRUE and meth.level = FALSE.

`jd.stat` logical(1). Whether to compute the JD statistic with asymptotic Chi-squared distribution with one degree of freedom (see `estimateJDiv`).

`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` logical(1) Whether methylation levels are given in place of counts. Default is FALSE.

`loss.fun` Loss function(s) used in the estimation of the best fitted model to beta distribution (only applied when Bayesian=TRUE; see (Loss function)). This fitting uses the approach followed in in the R package usefr. After $z = 1/2 * sum((f(x) - y)^2)$ we have:

1. "linear": linear function which gives a standard least squares: $loss(z) = z$.
2. "huber": Huber loss, $loss(z) = ifelse(z \leq 1, z, sqrt(z) - 1)$.
3. "smooth": Smooth approximation to the sum of residues absolute values: $loss(z) = 2 * (sqrt(z + 1) - 1)$.
4. "cauchy": Cauchy loss: $loss(z) = log(z + 1)$.
5. "arctg": arc-tangent loss function: $loss(x) = atan(z)$.

Loss 'linear' function works well for most of the methylation datasets with acceptable quality.

`logbase` Logarithm base used to compute the JD (if JD = TRUE). Logarithm base 2 is used as default (bit unit). Use logbase = $exp(1)$ for natural logarithm.

`verbose` if TRUE, prints the function log to stdout

`...` Optional parameters for `uniqueGRanges` function.

Details

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:

$$H = 2(n_1+1)(n_2+1) * ((sqrt(p_1) - sqrt(p_2))^2 + (sqrt(1-p_2) - sqrt(1-p_2))^2)/(n_1+n_2+2)$$

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.

According with the above equation, to estimate Hellinger divergence, not only the methylation levels are considered in the estimation of H, but also the control and treatment coverage at each given cytosine site. At this point, it is worthy to do mention that if the reference sample is derived with function `poolFromGRlist` using the 'sum' of read counts to compute a methylation pool, then 'min.coverage' parameter value must be used to prevent an over estimation of the divergence for low coverage cytosines sites. For example, if a reference sample is derived as the methylation pool of read count sum from 3 individuals and we want to consider only methylation sites with minimum coverage of 4, then we can set min.coverage = c(12, 4), where the number 12 (3 x 4) is the minimum coverage requested for the each cytosine site in the reference sample.

If the methylation levels are provided in place of counts, then the Hellinger divergence is computed as:

$$H = (sqrt(p_1) - sqrt(p_2))^2 + (sqrt(1 - p_1) - sqrt(1 - p_2))^2$$

This formula assumes that the probability vectors derived from the methylation levels $p_i = c(p_{i1}, 1 - p_{i2})$ (see `estimateHellingerDiv` are an unbiased estimation of the expected one. The function applies a pairwise filtering after building a single GRanges from the two GRanges objects. Experimentally available cytosine sites are paired using the function 'uniqueGRanges'.

It is important to observe that several filtering conditions are provided to select biological meaningful cytosine positions, which prevent to carry experimental errors in the downstream analyses. By filtering the read count we try to remove bad quality data, which would be in the edge of the experimental error originated by the BS-seq sequencing. It is user responsibility to check whether cytosine positions used in the analysis are biological meaningful. For example, a cytosine position with counts mC1 = 10 and uC1 = 20 in the 'ref' sample and mC2 = 1 and uC2 = 0 in an 'indv' sample will lead to methylation levels p1 = 0.333 and p2 = 1, respectively, and TV = p2 - p1 = 0.667, which apparently indicates a hypermethylated site. However, there are not enough reads supporting p2 = 1. A Bayesian estimation of TV will reveal that this site would be, in fact, hypomethylated. So, the best practice will be the removing of sites like that. This particular case is removed under the default settings: min.coverage = 4, min.meth = 4, and min.umeth = 0 (see example for function `uniqueGRfilterByCov`, called by 'estimateDivergence').

Value

An object from 'infDiv' class with the four columns of counts, the information divergence, and additional columns:

1) A matrix: The original matrix of methylated c_{ij} and unmethylated t_{ij} read counts from control $j = 1$ and treatment $j = 2$ samples at positions i.

2) 'p1' and 'p2': methylation levels for control and treatment, respectively. If 'meth.level = FALSE' and 'Bayesian = TRUE' (recommended), 'p1' and 'p2' are estimated following the Bayesian approach described in reference (1).

3) 'bay.TV': total variation TV = p2 - p1

4) 'TV': total variation based on simple counts: $TV = c_1/(c_1 + t_1) - c_2/(c_2 + t_2)$, where c_i and t_i denote methylated and unmethylated read counts, respectively.

5) Hellinger divergence, 'hdiv': If Bayesian = TRUE, the results are based on the posterior estimations of methylation levels. if meth.level = FALSE', then 'hdiv' is computed as given in reference (2), otherwise as:

$$hdiv = (sqrt(p_1) - sqrt(p_2))^2 + (sqrt(1 - p_1) - sqrt(1 - p_2))^2$$

Author(s)

Robersy Sanchez (https://genomaths.com)

References

1. Sanchez R, Yang X, Maher T, Mackenzie S. Discrimination of DNA Methylation Signal from Background Variation for Clinical Diagnostics. Int. J. Mol Sci, 2019, 20:5343.

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

See Also

`estimateBayesianDivergence`

Examples

```
## The read count data are created
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), JD = TRUE,
verbose = FALSE)[[1]]

## Keep in mind that Hellinger and J divergences are, in general,
## correlated
cor.test(x = as.numeric(hd$hdiv), y = as.numeric(hd$jdiv),
        method = 'kendall')
```

`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

The methylation level p_ij for an individual i at cytosine site j corresponds to a probability vector $p^ij = (p_ij, 1 - p_ij)$. Then, the information divergence between methylation levels p^1j and p^2j from individuals 1 and 2 at site j is the divergence between the vectors $p^1j = (p_1j, 1 - p_1j)$ and $p^2j = (p_2j, 1 - p_2j)$. If the vector of coverage is supplied, then the information divergence is estimated according to the formula:

$$hdiv = 2*(n_1+1)*(n_2+1)*((sqrt(p_1j)-sqrt(p_2j))^2+(sqrt(1-p_1j)-sqrt(1-p_2j))^2)/(n_1+n_2+2)$$

This formula corresponds to Hellinger divergence as given in the first formula from Theorem 1 from reference 1. Otherwise:

$$hdiv = (sqrt(p_1j) - sqrt(p_2j))^2 + (sqrt(1 - p_1j) - sqrt(1 - p_2j))^2$$

Missing methylation levels, reported as NA or NaN, are replaced with zero.

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)
```

`estimateJDiv` *J Information Divergence of Methylation Levels*

Description

Given a the methylation levels from two individuals at a given cytosine site, this function computes the J information divergence (*JD*) between methylation levels. The motivation to introduce *JD* in Methyl-IT is founded on:

1. It is a symmetrised form of Kullback–Leibler divergence (D_{KL}). Kullback and Leibler themselves actually defined the divergence as: $D_{KL}(P||Q)+D_{KL}(Q||P)$, which is symmetric and nonnegative, where the probability distributions *P* and *Q* are defined on the same probability space (see reference (1) and Wikipedia).
2. In general, *JD* is highly correlated with Hellinger divergence, which is the main divergence currently used in Methyl-IT (see examples for function `estimateDivergence`.
3. By construction, the unit of measurement of *JD* is: bit of information, which set the basis for further information-thermodynamics analyses.

Usage

```
estimateJDiv(
  p,
  logbase = 2,
  stat = FALSE,
  n = NULL,
  output = c("single", "all")
)
```

Arguments

p	A numerical vector of the methylation levels p = c(p1, p2) from individuals 1 and 2.
logbase	Logarithm base used to compute the JD. Logarithm base 2 is used as default. Use logbase = exp(1) for natural logarithm.
stat	logical(1). Whether to compute the statistic with asymptotic Chi-squared distribution with one degree of freedom (details).
output	Optional. If 'stat = TRUE', whether to return JD and its asymptotic chi-squared statistic.

Details

The methylation levels p_{ij}, at a given cytosine site *i* from an individual *j*, lead to the probability vectors $p^{ij} = (p_{ij}, 1 - p_{ij})$. Then, the J-information divergence between the methylation levels p_{ij} and the methylation levels q_i, used as reference, corresponds to the divergence between the probability vectors p^{ij} and $q^i = (q_i, 1 - q_i)$. *JD* is computed as in reference (1-2):

$$JD(p^{ij}, q^i) = ((p_{ij} - q_i) * log(p_{ij}/q_i) + (q_i - p_{ij}) * log((1 - p_{ij})/(1 - q_i)))/2$$

Missing methylation levels, reported as NA or NaN, are replaced with zero.

The statistic with asymptotic Chi-squared distribution is based on the statistic suggested by Kupperman (1957) (2) for D_{KL} and commented in references (3-4). That is,

$$2 * (n_1 + 1) * (n_2 + 1) * JD(p^{ij}, q^i)/(n_1 + n_2 + 2)$$

Where n_1 and n_2 are the total counts (coverage in the case of methylation) used to compute the probabilities p_{ij} and q_i. A basic Bayesian correction is added to prevent zero counts.

Value

The J divergence value for the given methylation levels is returned

Author(s)

Robersy Sanchez 11/27/2019 https://github.com/genomaths

References

1. Kullback S, Leibler RA. On Information and Sufficiency. Ann Math Stat. 1951;22: 79–86. doi:10.1214/aoms/1177729694.
2. Kupperman, M., 1957. Further application to information theory to multivariate analysis and statistical inference. Ph.D. Dissertation, George Washington University.
3. Salicrú M, Morales D, Menéndez ML, Pardo L. On the applications of divergence type measures in testing statistical hypotheses. Journal of Multivariate Analysis. 1994. pp. 372–391. doi:10.1006/jmva.1994.1068.
4. Basu A, Mandal A, Pardo L. Hypothesis testing for two discrete populations based on the Hellinger distance. Stat Probab Lett. Elsevier B.V.; 2010;80: 206–214. doi:10.1016/j.spl.2009.10.008.
5. J. K. Chung, P. L. Kannappan, C. T. Ng, P. K. Sahoo, Measures of distance between probability distributions. J. Math. Anal. Appl. 138, 280–292 (1989).
6. Lin J. Divergence Measures Based on the Shannon Entropy. IEEE Trans Inform Theory, 1991, 37:145–51.
7. Sanchez R, Mackenzie SA. Information thermodynamics of cytosine DNA methylation. PLoS One, 2016, 11:e0150427.

See Also

https://en.wikipedia.org/wiki/Kullback-Leibler_divergence for more details, and `estimateDivergence` for an example of using it.

Examples

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

## A numerical trick is implemented. The J-divergence values are the
## same for the following vectors:
p <- c(0.9999999999999999, 0)
q <- c(1, 0)

estimateJDiv(p) == estimateJDiv(q)
```

`evaluateDIMPclass` *Evaluate DMPs Classification*

Description

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

Usage

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

Arguments

LR	An object from 'pDMP' class. including control and treatment GRanges containing divergence values for each DMP 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 the 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', TV estimated with Bayesian correction of methylation levels 'bay.TV', probability of potential DMP '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, respectively. If 'wprob = TRUE', then Logarithm base-10 of 'wprob' will be used as predictor in place of 'wprob'.
classifier	Classification model to use. Option 'logistic' applies a logistic regression model; option 'lda' applies a Linear Discriminant Analysis (LDA); 'qda' applies a Quadratic Discriminant Analysis (QDA), 'pca.logistic' applies logistic regression model using the Principal Component (PCs) estimated with Principal Component Analysis (PCA) as predictor variables. pca.lda' applies LDA using PCs as predictor variables, and the option 'pca.qda' applies a Quadratic Discriminant Analysis (QDA) using PCs as predictor variables.

`pval.col` Column number for p-value used in the performance analysis and estimation of the cutpoints. Default: NULL. If NULL it is assumed that the column is named 'wprob'.

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

`center` A logical value indicating whether the variables should be shifted to be zero centered (same as in 'prcomp' prcomp). Only used if classifier = 'pcaLDA'.

`scale` A logical value indicating whether the variables should be scaled to have unit variance before the analysis takes place (same as in 'prcomp' prcomp). Only used if classifier = 'pcaLDA'.

`interactions` 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 DMPs) 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 DMPs).

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

`seed` Random seed used for random number generation.

`verbose` if TRUE, prints the function log to stdout

Details

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

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

Value

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

Examples

```
## Get a data set of DMPs
data(dmps, package = 'MethylIT')

## Classification of DMPs into two clases: DMPS from control and DMPs
## from treatment samples and evaluation of the classifier performance
## (for more details see ?evaluateDIMPclass).
perf <- evaluateDIMPclass(LR = dmps,
                          column = c(hdiv = TRUE, TV = TRUE,
                                     wprob = TRUE, pos = TRUE),
                          classifier = 'lda', n.pc = 4L,
                          control.names =  c('C1', 'C2', 'C3'),
                          treatment.names = c('T1', 'T2', 'T3'),
                          center = TRUE, scale = TRUE, prop = 0.6)

## Model classification performance
perf$Performance
```

---

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 GRanges 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 metacolumn table or the columns with methylated (mC) and unmethylated counts (uC).

Value

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

Examples

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

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

Description

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

Usage

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

Arguments

`LR`	A list of GRanges, a GRangesList, a CompressedGRangesList object, or an object from Methyl-IT downstream analyses: 'InfDiv' or 'pDMP' object. Each GRanges object from the list must have two columns: methylated (mC) and unmethylated (uC) counts. The name of each element from the list must coincide with a control or a treatment name.
`count.col`	2d-vector of integers with the indexes of the read count columns. If not given, then it is assumed that the methylated and unmethylated read counts are located in columns 1 and 2 of each GRanges metacolumns. If object LR is the output of Methyl-IT function `estimateDivergence`, then columns 1:4 are the read count columns: columns 1 and 2 are methylated and unmethylated read counts from the reference group, while columns 3 and 4 are methylated and unmethylated read counts from the treatment group, respectively. In this case, if the requested comparison is reference versus treatment, then no specification is needed for count.col. The comparison control versus treatment can be obtained by setting count.col = 3:4 and providing control.names and treatment.names.

`control.names, treatment.names`
: Names/IDs of control and treatment samples, which must be included in the variable GR at the metacolumn. Default NULL. If provided the Fisher's exact test control versus treatment is performed. Default is NULL. If NULL, then it is assumed that each GRanges object in LR has four columns of counts. The first two columns correspond to the methylated and unmethylated counts from control/reference and the other two columns are the methylated and unmethylated counts from treatment, respectively.

`pooling.stat`
: statistic used to estimate the methylation pool: row sum, row mean or row median of methylated and unmethylated read counts across individuals. If the number of control samples is greater than 2 and pooling.stat is not NULL, then they will pooled. The same for treatment. Otherwise, all the pairwise comparisons will be done.

`tv.cut`
: A cutoff for the total variation distance (TVD; absolute value of methylation levels differences) estimated at each site/range as the difference of the group means of methylation levels. If tv.cut is provided, then sites/ranges k with $|TV_k| < tv.cut$ are removed before performing the regression analysis. Its value must be NULL or a number $0 < tv.cut < 1$.

`hdiv.cut`
: An optional cutoff for the Hellinger divergence (*hdiv*). If the LR object derives from the previous application of function `estimateDivergence`, then a column with the *hdiv* values is provided. If combined with tv.cut, this permits a more effective filtering of the signal from the noise. Default is NULL.

`hdiv.col`
: Optional. Columns where *hdiv* values are located in each GRanges object from LR. It must be provided if together with *hdiv.cut*. Default is NULL.

`pAdjustMethod`
: method used to adjust the results; default: BH

`pvalCutOff`
: cutoff used then a p-value adjustment is performed

`saveAll`
: if TRUE all the temporal results are returned

`num.cores`
: The number of cores to use, i.e. at most how many child processes will be run simultaneously (see bpapply function from BiocParallel).

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

`verbose`
: if TRUE, prints the function log to stdout

`progressbar`
: logical(1). Enable progress bar

`...`
: Additional parameters for function `uniqueGRanges`.

Details

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

Value

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

See Also

rmstGR

Examples

```
## Get a dataset of Hellinger divergency of methylation levels
## from the package
data(HD)

### --- To get the read counts
hd <- lapply(HD, function(hd) {
hd = hd[1:10,3:4]
colnames(mcols(hd)) <- c('mC', 'uC')
return(hd)
})

FisherTest(LR = hd, pooling.stat = NULL, control.names = 'C1',
treatment.names = 'T1', pAdjustMethod='BH', pvalCutOff = 0.05,
num.cores = 1L, verbose=FALSE)
```

---

fitGammaDist *Nonlinear fit of Gamma CDF (Gamma)*

Description

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

Usage

```
fitGammaDist(
  x,
  probability.x,
  parameter.values,
  location.par = FALSE,
  sample.size = 20,
  npoints = NULL,
  maxiter = 1024,
  ftol = 1e-12,
  ptol = 1e-12,
  maxfev = 1e+05,
  nlms = FALSE,
  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 parameter is included (mu != 0), this must be given as parameter.values = list(shape = 'value', scale = 'value', mu = 'value') or if mu = 0, as: parameter.values = list(shape = 'value', scale = 'value'). If not provided, then an initial guess is provided.
location.par	whether to consider the fitting to generalized gamma distribution (Gamma) including the location parameter, i.e., a Gamma with four parameters (GGamma3P).
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.
nlms	Logical. Whether to return the nonlinear model object nls.lm. Default is FALSE.
verbose	if TRUE, prints the function log to stdout

Details

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

If the number of values to fit is >10^6, the fitting to a GGamma CDF would be a time consuming task. To reduce the computational time, the data can be 'summarized' into 'npoints' (bins) and used as the new predictors.

Value

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

Author(s)

Robersy Sanchez - 06/03/2016

References

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

Examples

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

---

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

Description

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

Usage

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

Arguments

x	numerical vector
parameter.values	initial parameter values for the nonlinear fit. If the locator parameter 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 (GGamma4P).
sample.size	Minimum size of the sample.
npoints	number of points used in the fit. If the number of points if greater than 10^6, then the fit is automatically set to npoints = 999999. However, the reported values for R.Cross.val, AIC, and BIC are computed taking into account the whole set of points.
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.
nlms	Logical. Whether to return the nonlinear model object nls.lm. Default is FALSE.
verbose	if TRUE, prints the function log to stdout
...	arguments passed to or from other methods.

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 npoint != NULL, the original variable values are summarized into 'npoint' bins and their midpoints are used as the new predictors. In this case, only the goodness-of-fit indicators AIC and R.Cross.val are estimated based on all the original variable x values.

Value

Model table with coefficients and goodness-of-fit results: Adj.R.Square, deviance, AIC, R.Cross.val, and rho, as well as, the coefficient covariance matrix. If **nlms = TRUE**, then a list with nonlinear model object nls.lm is returned.

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(123)
## Fitting GGamma3P
x <- rggamma(3000, alpha = 1.03, psi = 0.75, scale = 1.1)
fitGGammaDist(x)
## Fitting GGamma4P
x <- x + 1
fitGGammaDist(x, location.par = TRUE)
```

`fitLogNormDist` *Nonlinear fit of Log-Normal CDF (LogNorm)*

Description

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

Usage

```
fitLogNormDist(
  x,
  probability.x,
  parameter.values,
  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 parameter is included (mu != 0), this must be given as parameter.values = list(alpha = 'value', scale = 'value', mu = 'value') or if mu = 0, as: parameter.values = list(alpha = 'value', scale = 'value'). If not provided, then an initial guess is provided.

`summarized.data`
: Logic value. If TRUE (default: FALSE), summarized data based on 'npoints' are used to perform the nonlinear fit.

`sample.size`
: size of the sample.

`npoints`
: number of points used in the fit.

`maxiter`
: positive integer. Termination occurs when the number of iterations reaches maxiter. Default value: 1024.

`ftol`
: non-negative numeric. Termination occurs when both the actual and predicted relative reductions in the sum of squares are at most ftol. Therefore, ftol measures the relative error desired in the sum of squares. Default value: 1e-12

`ptol`
: non-negative numeric. Termination occurs when the relative error between two consecutive iterates is at most ptol. Therefore, ptol measures the relative error desired in the approximate solution. Default value: 1e-12.

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

`verbose`
: if TRUE, prints the function log to stdout

Details

The algorithm tries to fit the two-parameter LogNorm CDF 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 LogNorm CDF would be a time consuming task. To reduce the computational time, the option 'summarized.data' can be set 'TRUE'. If summarized.data = TRUE, the original variable values are summarized into 'npoint' bins and their midpoints are used as the new predictors. In this case, only the goodness-of-fit indicators AIC and R.Cross.val are estimated based on all the original variable x values.

Value

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

Author(s)

Robersy Sanchez - 04/09/2019

References

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

Examples

```
set.seed(126)
x <- rlnorm(1000, meanlog = 1.03, sdlog = 2.1)
fitLogNormDist(x)
```

---

getDIMPatGenes *Count DMPs at gene-body*

Description

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

Usage

```
getDIMPatGenes(
  GR,
  GENES,
  type = "within",
  ignore.strand = TRUE,
  only.hypo = FALSE,
  only.hyper = FALSE,
  output = c("list", "GRanges"),
  by.coord = FALSE,
  ...
)

## Default S3 method:
getDIMPatGenes(
  GR,
  GENES,
  type = "within",
  ignore.strand = TRUE,
  only.hypo = FALSE,
  only.hyper = FALSE,
  output = NULL,
  by.coord = FALSE,
  ...
)

## S3 method for class 'GRanges'
getDIMPatGenes(
  GR,
  GENES,
  type = "within",
  ignore.strand = TRUE,
```

```
  only.hypo = FALSE,
  only.hyper = FALSE,
  output = NULL,
  by.coord = FALSE,
  ...
)

## S3 method for class 'pDMP'
getDIMPatGenes(
  GR,
  GENES,
  type = "within",
  ignore.strand = TRUE,
  only.hypo = FALSE,
  only.hyper = FALSE,
  output = c("list", "GRanges"),
  by.coord = FALSE,
  ...
)

## S3 method for class 'InfDiv'
getDIMPatGenes(
  GR,
  GENES,
  type = "within",
  ignore.strand = TRUE,
  only.hypo = FALSE,
  only.hyper = FALSE,
  output = c("list", "GRanges"),
  by.coord = FALSE,
  ...
)

## S3 method for class 'list'
getDIMPatGenes(
  GR,
  GENES,
  type = "within",
  ignore.strand = TRUE,
  only.hypo = FALSE,
  only.hyper = FALSE,
  output = c("list", "GRanges"),
  by.coord = FALSE,
  ...
)
```

Arguments

GR	An objects object from the any of the classes: 'pDMP', 'InfDiv', GRangesList, GRanges or a list of GRanges.
GENES	A GRanges object with gene coordinates and gene IDs. A column named **'gene_id'** carrying the gene ids should be included in the metacolumns. If the meta-column named 'gene_id' is not provided, then gene (region) ids will be created using the gene (region) coordinates.
ignore.strand, type	Same as for findOverlaps-methods.
only.hypo, only.hyper	logical(1). Whether to select only hypo-methylated or hyper-methylated cytosine sites.
output	Class of the object to be returned, a "list", or a "GRanges" object.
by.coord	logical(1). If TRUE, then the DMP are count per coordinate and not per gene id.
...	optional arguments for findOverlaps-methods. Users must evaluate whether specific setting makes sense on each particular context.

Details

If ***by.coord == FALSE*** and "gene_id" is provided in ***GENES*** argument, then DMP counts are made per gene-id. Hence, DMPs from different regions with the same gene-id, say e.g. exons, will be pooled in the count as they bear the same id.

Value

A a list GRanges object.

See Also

getDMPatRegions

Examples

```
## 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'))

## Get a dataset of potential signals and the estimated cutpoint from the
## package
data(PS, cutpoint)

## The estimated cutpoints are used to discriminate signals from the noise.
## That is, DMPs are selected using the cupoints
DIMPs <- selectDIMP(PS, div.col = 9L, cutpoint = cutpoint$cutpoint)

## Finally DMPs found on genes
```

```
DIMR <- getDIMPatGenes(GR = DIMPs$T1, GENES = genes)
```

`getDMPatRegions` *Count DMPs at Genomic Regions*

Description

The function counts DMPs overlapping with genomic regions. In fact, this function operates as `getDIMPatGenes` function, but without the restrictions set for GRanges objects derived from MethylIT pipeline.

Usage

```
getDMPatRegions(
  GR,
  regions,
  only.hypo = FALSE,
  only.hyper = FALSE,
  type = "within",
  ignore.strand = TRUE,
  ...
)

## Default S3 method:
getDMPatRegions(
  GR,
  regions,
  only.hypo = FALSE,
  only.hyper = FALSE,
  type = "within",
  ignore.strand = TRUE,
  ...
)

## S3 method for class 'GRanges'
getDMPatRegions(
  GR,
  regions,
  only.hypo = FALSE,
  only.hyper = FALSE,
  type = "within",
  ignore.strand = TRUE,
  ...
)

## S3 method for class 'pDMP'
```

```
getDMPatRegions(
  GR,
  regions,
  only.hypo = FALSE,
  only.hyper = FALSE,
  type = "within",
  ignore.strand = TRUE,
  ...
)

## S3 method for class 'InfDiv'
getDMPatRegions(
  GR,
  regions,
  only.hypo = FALSE,
  only.hyper = FALSE,
  type = "within",
  ignore.strand = TRUE,
  ...
)

## S3 method for class 'list'
getDMPatRegions(
  GR,
  regions,
  only.hypo = FALSE,
  only.hyper = FALSE,
  type = "within",
  ignore.strand = TRUE,
  ...
)
```

Arguments

GR	An objects object from the any of the classes: 'pDMP', 'InfDiv', GRangesList, GRanges or a list of GRanges.
regions	A GRanges object with gene coordinates and gene IDs. A meta-column named 'gene_id' carying the gene ids should be included. If the meta-column named 'gene_id' is not provided, then gene (region) ids will be created using the gene (region) coordinates.
only.hypo, only.hyper	logical(1). Whether to select only hypo-methylated or hyper-methylated cytosine sites.
ignore.strand, type	Same as for findOverlaps-methods
...	optional arguments for findOverlaps-methods. Users must evaluate whether specific setting makes sense on each particular context.

Value

A GRanges object

See Also

getDIMPatGenes

Examples

```
## 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'))

## Get a dataset of dmps from the package
data(dmps)

## Finally DMPs found on genes
dmrs <- getDMPatRegions(GR = dmps, regions = 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.

## Not run:
 filenames <- getGEOSuppFiles(GEO = 'GSM881757',
                pattern = 'G_cytosine.txt.gz')

 file.remove(filenames) ## Remove the downloaded file

## End(Not run)
```

---

getPotentialDIMP *Potential methylation signal*

Description

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

Usage

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

Arguments

LR	An object from 'InfDiv' or 'testDMP' class. These objects are previously obtained with function estimateDivergence or FisherTest.
nlms	A list of distribution fitted models (output of gofReport function) or NULL. If NULL, then empirical cumulative distribution function is used to get the potential DMPs.
div.col	Column number for divergence variable is located in the meta-column.
dist.name	Name of the fitted distribution. This could be the name of one distribution or a characters vector of length(nlms). Default is the two parameters Weibull distribution: 'Weibull2P'. The available options are: **"Weibull2P"** Weibull with two-parameters. **"Weibull3P"** Weibull with three-parameters. **"Gamma2P"** Gamma with two-parameters. **"Gamma3P"** Gamma with three-parameters. **"GGamma3P"** Generalized gamma with three-parameters. **"GGamma4P"** Generalized gamma with four-parameters. **"ECDF"** The empirical cumulative distribution function. **"None"** No distribution. If **dist.name != 'None'**, and **nlms != NULL**, then a column named 'wprob' with a probability vector derived from the application of model 'nlms' will be returned.
absolute	Logic (default, FALSE). Total variation (TV, the difference of methylation levels) is normally an output in the downstream MethylIT analysis. If 'absolute = TRUE', then TV is transformed into |TV|, which is an information divergence that can be fitted to Weibull or to Generalized Gamma distribution. So, if the nonlinear fit was performed for |TV|, then absolute must be set to TRUE.

`alpha` A numerical value (usually $\alpha \leq 0.05$) used to select cytosine sites k with information divergence (DIV_k) for which the the probabilities hold: $P(DIV_k > DIV(\alpha))$.

`pval.col` An integer denoting a column from each GRanges object from LR where p-values are provided when **dist.name == 'None'** and **nlms == NULL**. Default is NULL. If NUll and **dist.name == 'None'** and **nlms == NULL**, then a column named **adj.pval** will used to select the potential DMPs.

`tv.col` Column number for the total variation to be used for filtering cytosine positions (if provided).

`tv.cut` If tv.cut and tv.col are provided, then cytosine sites k with $abs(TV_k) < tv.cut$ are removed before to perform the ROC analysis.

`min.coverage` Cytosine sites with coverage less than min.coverage are discarded. Default: 0

`hdiv.col` Optional. A column number for the Hellinger distance to be used for filtering cytosine positions. Default is NULL.

`hdiv.cut` If hdiv.cut and hdiv.col are provided, then cytosine sites k with hdiv < hdiv.cut are removed.

`pAdjustMethod`
method used to adjust the p-values from other approaches like Fisher's exact test, which involve multiple comparisons Default is NULL. Do not apply it when a probability distribution model is used (**when nlms is given**), since it makes not sense.

Details

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

Value

A list of GRanges objects, each GRanges object carrying the selected cytosine sites and the probabilities that the specified divergence values can be greater than the critical value specified by α: $P(DIV_k > DIV(\alpha))$.

Examples

```
## Get a dataset of Hellinger divergency of methylation levels and their
## corresponding best nonlinear fit distribution models from the package
data(HD, gof)
PS <- getPotentialDIMP(LR = HD, nlms = gof$nlms, dist.name = gof$bestModel,
                    div.col = 9L, alpha = 0.05)
```

ggamma *Generalized Gamma distribution*

Description

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

Usage

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

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

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

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

Arguments

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

p	vector of probabilities.
n	number of observations.

Details

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

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

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

Value

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

References

1. Handbook on STATISTICAL DISTRIBUTIONS for experimentalists (p. 73) by Christian Walck. Particle Physics Group Fysikum. University of Stockholm (e-mail: walck@physto.se)
2. Stacy, E. W. A Generalization of the Gamma Distribution. Ann. Math. Stat. 33, 1187–1192 (1962).
3. Stacy E, Mihram G (1965) Parameter estimation for a generalized gamma distribution. Technometrics 7: 349-358.
4. Sanchez, R. & Mackenzie, S. A. Information Thermodynamics of Cytosine DNA Methylation. PLoS One 11, e0150427 (2016).

Examples

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

## To fit random generated numbers
set.seed(123)
x <- rggamma(2000, alpha = 1.03, psi = 0.75, scale = 2.1)
fitGGammaDist(x)
```

glmDataSet *Data set constructor for class glmDataSet*

Description

This function is used to build a object suitable to be used with Methyl-IT `countTest2` function.

Usage

```
glmDataSet(GR = NULL, counts = NULL, colData = NULL)
```

Arguments

GR	A `GRanges-class` object with the count matrix of DMPs in the metacolumns (see *'counts'*). If provided, then leave parameter *'counts = NULL'*.
counts	Count matrix of DMPs with minimal dimensions 1 (row) x 4 (columns). Column names must corresponds to the rownames from parameter 'colData'.
colData	A data frame with one columnn named 'condition', which must be a factor with exactly two levels. The rownames of *colData* individual samples. The row names of *colData* must correspond to th column names of the count matrix.

Details

Data set constructor for class glmDataSet also validate the object

Value

An object from 'RangedGlmDataSet' class containing these attributes: 'GR': the GRanges of the object, 'counts': the counts for each sample at that genomic position, 'colData': the condition of each sample, treatment or control, 'sampleNames': the names of the samples, 'levels': the values (perhaps TT and CT for treatment and control) permitted in the 'colData' attribute, optionData: additional metadata or NULL

Author(s)

Robersy Sanchez

Examples

```
set.seed(133) # Set a seed
## A GRanges object with the count matrix in the metacolumns is created
countData <- matrix(sample.int(200, 500, replace = TRUE), ncol = 4)
colnames(countData) <- c('A1','A2','B1','B2')
start <- seq(1, 25e4, 2000)
end <- start + 1000
chr <- c(rep('chr1', 70), rep('chr2', 55))
GR <- GRanges(seqnames = chr, IRanges(start = start, end = end))
mcols(GR) <- countData
```

```
## Gene IDs
names(GR) <- paste0('gene', 1:length(GR))

## An experiment design is set.
colData <- data.frame(condition = factor(c('A','A','B','B')),
c('A1','A2','B1','B2'), row.names = 2)

## A RangedGlmDataSet is created
glmDataSet(GR = GR, colData = colData)
```

`gof` *Simulated dataset of nonlinear fits used in the examples*

Description

Each individuals sample includes 10000 cytosine positions

Usage

```
gof
```

Format

gof is a list of best fitted nonlinear probability distribution model estimated for the dataset 'HD' (available in MethylIT package) 'gof' carries the information on the best fitted probability distribution model for each individual sample. 'gof' was obtained with function `gofReport`.

`HD` *Simulated dataset of Hellinger divergences used in the examples*

Description

Each individuals sample includes 10000 cytosine positions

Usage

```
HD
```

Format

HD is an object from class 'InfDiv' carrying in the meta-columns the following variables:

p1 methylation level from the reference sample

p2 methylation level from the treatment sample

TV the total variation distance (difference of methylation levels)

hdiv Hellinger divergence

```
'HD' was obtained with function \code{\link{estimateDivergence}}.
```

`lda_perf` *Classification LDA model for simulated dataset of DMPs used in examples*

Description

This data/object carries the information about the classification performance of a Linear Discriminant (*"LDA"*)) model on the set of `dmps`.

Usage

```
lda_perf
```

Format

lda_perf is list object consisting of the following elements:

Performance Classification performance of the *"LDA"* model on the set of `dmps`.

FDR False discovery rate estimated for the *"LDA"* model on the set of `dmps`

model The `lda` model fitted on the set of `dmps`.

lda_perf is a list object obtained applying `evaluateDIMPclass` function on the set of `dmps`.

`logit_perf` *Classification logistic model for simulated dataset of DMPs used in examples*

Description

This data/object carries the information about the classification performance of *"logistic"* model on the set of `dmps`.

Usage

```
logit_perf
```

Format

logit_perf is an object from class *"LogisticR"*, consisting of a list with the following elements:

Performance Classification performance of the *"logistic"* model on the set of `dmps`.

FDR False discovery rate estimated for the logistic model on the set of `dmps`

model The fitted `glm` logistic model fitted on the set of `dmps`.

logit_perf is an object from "LogisticR" class, which was obtained applying `evaluateDIMPclass` function on the set of `dmps`.

`MethylIT` *MethylIT: Methylation Analysis Based on Signal Detection and Machine Learning*

Description

Methyl-IT implements methylation analysis based on signal detection theory and machine-learning. Methylation changes are expressed in terms of information divergence (ID) of methylation level. Methylation process is a stochastic process, particularly, a biochemical-biophysical process which must not be reduced to statistic. Methyl-IT includes the information on the statistical biophysics of the methylation process with the estimation of the probability distribution of the methylation background noise (plus signal), which is used as null distribution for the application of basic signal detection theory.

meth_levels *Compute methylation levels*

Description

This function computes the

Usage

```
meth_levels(
  GR,
  x,
  columns = c(mC1 = 1, uC1 = 2, mC2 = NULL, uC2 = NULL),
  Bayesian = FALSE,
  init.pars = NULL,
  min.coverage = 4,
  tv = FALSE,
  bay.tv = FALSE,
  filter = FALSE,
  preserve.dt = FALSE,
  loss.fun = c("linear", "huber", "smooth", "cauchy", "arctg"),
  num.cores = 1,
  tasks = 0L,
  verbose = TRUE,
  ...
)

## S4 method for signature 'ANY,data.frame'
meth_levels(
  GR,
  x,
  columns = c(mC1 = 1, uC1 = 2, mC2 = NULL, uC2 = NULL),
  Bayesian = FALSE,
  init.pars = NULL,
  min.coverage = 4,
  tv = FALSE,
  bay.tv = FALSE,
  filter = FALSE,
  preserve.dt = FALSE,
  loss.fun = c("linear", "huber", "smooth", "cauchy", "arctg"),
  num.cores = 1,
  tasks = 0L,
  verbose = TRUE,
  ...
)

## S4 method for signature 'GRanges,ANY'
```

```
meth_levels(
  GR,
  x,
  columns = c(mC1 = 1, uC1 = 2, mC2 = NULL, uC2 = NULL),
  Bayesian = FALSE,
  init.pars = NULL,
  min.coverage = 4,
  tv = FALSE,
  bay.tv = FALSE,
  filter = FALSE,
  preserve.dt = FALSE,
  loss.fun = c("linear", "huber", "smooth", "cauchy", "arctg"),
  num.cores = 1,
  tasks = 0L,
  verbose = TRUE,
  ...
)

## S4 method for signature 'list,ANY'
meth_levels(
  GR,
  x = NULL,
  columns = c(mC1 = 1, uC1 = 2, mC2 = 0, uC2 = 0),
  Bayesian = FALSE,
  init.pars = NULL,
  min.coverage = 4,
  tv = FALSE,
  bay.tv = FALSE,
  filter = FALSE,
  preserve.dt = FALSE,
  loss.fun = c("linear", "huber", "smooth", "cauchy", "arctg"),
  num.cores = 1,
  tasks = 0L,
  verbose = TRUE,
  ...
)
```

Arguments

GR, x	A GRanges-class object (***GR***) or 'data.frame' (***x***) with a matrix of counts in the meta-columns (methylated mC and unmethylated uC cytosines) or a list of GRanges-class objects.
columns	Vector of one or two integer numbers denoting the indexes of the columns where the methylated and unmethylated read counts are found. Unless specified in the parameter 'columns', the methylation counts must be given in the first four columns: 'mC1' and 'uC1' methylated and unmethylated counts for control sample, and 'mC2' and 'uC2' methylated and unmethylated counts for treatment sample, respectively.

Bayesian	logical(1). Whether to perform the estimations based on posterior estimations of methylation levels.
init.pars	initial parameter values. Defaults is NULL and an initial guess is estimated using optim function. If the initial guessing fails initial parameter values are to alpha = 1 & beta = 1, which imply the parsimony pseudo-counts greater than zero.
min.coverage	An integer or an integer vector of length 2. Cytosine sites where the coverage in both samples, 'x' and 'y', are less than min.coverage' are discarded. The cytosine site is preserved, however, if the coverage is greater than 'min.coverage' in at least one sample. If 'min.coverage' is an integer vector, then the corresponding min coverage is applied to each sample.
tv	logical(1). Whether to compute the total variation distance at each cytosine site. That is, the difference of methylation levels.
bay.tv	logical(1). Whether to compute the total variation distance at each cytosine site based on Bayesian estimation of methylation levels.
filter	logical(1). Optional. If TRUE, then only cytosine sites with $coverages > min.coverage$ are including in the computation.
preserve.dt	logical(1). Option of whether to preserve all the metadata from the original 'data.frame' or GRanges-class object.
loss.fun	Described in estimateBetaDist.
num.cores, tasks	Parameters for parallel computation using package BiocParallel-package: the number of cores to use, i.e. at most how many child processes will be run simultaneously (see bplapply and the number of tasks per job (only for Linux OS). These parameters will be passed to uniqueGRanges.
verbose	if TRUE, prints the function log to stdout
...	Optional parameter values for: maxiter, ftol, ptol, and gradtol from nlsLM and nlm functions.

Author(s)

Robersy Sanchez (https://genomaths.com)

Examples

```
## The read count data are created
num.samples <- 250
s <- 1:num.samples
gr <- data.frame(chr = 'chr1', start = s, end = s,
                 strand = sample(c("+", "-"), num.samples, replace = TRUE),
                 mCc = rnbinom(size = num.samples, mu = 4, n = 500),
                 uCc = rnbinom(size = num.samples, mu = 4, n = 500),
                 mCt = rnbinom(size = num.samples, mu = 4, n = 500),
                 uCt = rnbinom(size = num.samples, mu = 4, n = 500))

gr <- makeGRangesFromDataFrame(gr, keep.extra.columns = TRUE)
```

```
gr <- meth_levels(GR = gr,
                  columns = c(mC1 = 1, uC1 = 2,
                              mC2 = 3, uC2 = 4),
                  preserve.dt = TRUE,
                  Bayesian = TRUE, tv = TRUE, bay.tv = TRUE,
                  num.cores = 1)
```

`nonlinearFitDist` *Nonlinear fit of Information divergences distribution*

Description

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

Usage

```
nonlinearFitDist(
  LR,
  column = 9,
  dist.name = "Weibull",
  sample.size = 20,
  location.par = FALSE,
  absolute = FALSE,
  npoints = NULL,
  model = "all",
  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(s) of the distribution to fit. A single character string or character vector naming the distribution(s): 'Weibull' (default), gamma with three-parameter (Gamma3P), gamma with two-parameter (Gamma2P), generalized gamma with three-parameter ('GGamma3P') or four-parameter ('GGamma4P'), and Log-Normal (LogNorm).

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 (GGamma4P).
absolute	Logic (default, FALSE). Total variation (TV, the difference of methylation levels) is normally an output in the downstream MethylIT analysis. If 'absolute = TRUE', then TV is transformed into |TV|, which is an information divergence that can be fitted to Weibull or to Generalized Gamma distribution.
npoints	number of points to be used in the fit. Default is NULL.
model	Optional. Only when dist.name = 'Weibull'. A selection of the distribution model, two-parameters and three-parameters Weibull model ('2P' and '3P'). Default is 'all' and the model with the best AIC criterion is reported. Alternatively, just use dist.name = 'Weibull2P' or dist.name = 'Weibull3P'.
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. 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-class from BiocParallel package).
maxfev	integer; termination occurs when the number of calls to fn has reached maxfev. Note that nls.lm sets the value of maxfev to 100*(length(par) + 1) if maxfev = integer(), where par is the list or vector of parameters to be optimized.
verbose	If TRUE, prints the function log to stdout
...	other parameters

Details

The algorithm prepares the information divergence variable to try fitting Weibull or generalized gamma distribution model to the data. If Weibull distribution is selected (default: 'Weibull'), function 'Weibull2P' first attempts fitting to the two-parameter Weibull CDF (Weibull2P). If Weibull2P

did not fit, then the algorithm will try to fit Weibull3P. The Levenberg-Marquardt algorithm implemented in R package 'minpack.lm' is used to perform the nonlinear fit. Cross-validations for the nonlinear regressions (R.Cross.val) are performed in each methylome as described in reference (1-2). In addition, Stein's formula for adjusted R squared (rho) is used as an estimator of the average cross-validation predictive power (2).

If 'GGamma3P' is selected the call to function 'fitGGammaDist' permits the fitting to the three-parameter GGamma CDF ('GGamma3P'). The fit to the four-parameter GGamma ('GGamma4P') is also available. GGamma distribution are fitted using a modification of Levenberg-Marquardt algorithm implemented in function 'nls.lm' from the 'minpack.lm' R package. Notice that the fit to GGamma distribution 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 https://github.com/genomaths

References

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

See Also

gofReport

Examples

```
## Load a dataset with Hellinger Divergence of methylation levels on it.
data(HD)

## The nonlinear fit based on three-parameter GGamma distribution
nlms2 <- nonlinearFitDist(HD, npoints = 100, dist.name = 'GGamma3P',
                          verbose = FALSE)

## Weilbull distribution is a particular case of GGamma.
nlms <- nonlinearFitDist(HD, npoints = 100, verbose = FALSE)

## The goodness-of-fit indicators AIC suggests that the best fitted model
## is obtained with GGamma distribution (in this example).
res <- mapply(function(m1,m2) as.numeric(c(Weibull = m1$AIC[1],
                                           GGamma = m2$AIC[1])),
              nlms, nlms2)
rownames(res) <-c('Weibull', 'GGamma')
```

```
res

## However, the Cross-validations correlation coefficient is saying that
## the Weibull distribution would be a little better probability
## predictor.
res <- mapply(function(m1,m2) as.numeric(c(Weibull = m1$R.Cross.val[1],
                                           GGamma = m2$R.Cross.val[1])),
              nlms, nlms2)
rownames(res) <-c('Weibull', 'GGamma')
res
```

`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 = 1e-04,
  method = "moment",
  max.pc = NULL,
  columns = 9L,
  ...
)

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

Arguments

formula	Same as in 'lda'from package 'MASS'.
data	Same as in 'lda'from package 'MASS' or an object from "pDMP" or "InfDiv" class.
grouping	Same as in 'lda' from package 'MASS'.
n.pc	Number of principal components to use in the LDA.
scale	Same as in 'prcomp' from package 'prcomp'.
center	Same as in 'prcomp' from package 'prcomp'.
tol	Same as in 'prcomp' from package 'prcomp'.
method	Same as in 'lda'from package 'MASS'.
max.pc	Same as in parameter 'rank.' from package 'prcomp'.
columns	Optional. Only used if 'data' belong to the "pDMP" or "InfDiv" class. Default is 9L.
...	Not in use.
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' basic predictions: classes and posterior classification probabilities. Option 'lda.pred' returns the object given by function 'predict.lda' from MASS package: 'class', 'posterior', 'scores' (cases scores on discriminant variables, see lda.

Details

The principal components (PCs) are obtained using the function 'prcomp' from R package 'stats', while the LDA is performed using the 'lda' function from R package 'MASS'. The current application only uses 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.

See Also

pcaQDA, lda and predict.lda

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)
```

`pcaLda_perf` *Classification PCA+LDA model for simulated dataset of DMPs used in examples*

Description

This data/object carries the information about the classification performance of the combined models of Principal Components *PCA* and Linear Discriminant (*"LDA"*)) analyses on the set of `dmps`.

Usage

```
pcaLda_perf
```

Format

pcaLda_perf is an object from class *"pcaLDA"*, consisting of a list with the following elements:

Performance Classification performance of the *"PCA+LDA"* model on the set of `dmps`.

FDR False discovery rate estimated for the *"PCA+LDA"* model on the set of `dmps`.

model The *"PCA+LDA"* model fitted on the set of `dmps`, carrying the `lda` and the *"PCA"* models. The *"PCA"* is fitted with `prcomp`function.

pcaLda_perf is an object from "pcaLDA" class, which was obtained applying `evaluateDIMPclass` function on the set of `dmps`.

`pcaLogisticR` *Logistic Classification Model using Principal Component Analysis (PCA)*

Description

Principal components (PCs) are estimated from the predictor variables provided as input data. Next, the individual coordinates in the selected PCs are used as predictors in the logistic regression.

Logistic regression using Principal Components from PCA as predictor variables

Usage

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

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

Arguments

`formula`	Same as in 'glm' from package 'stats'. One term carrying interaction between two variables can be introduced (with notation as indicated in `formula` function).
`data`	Same as in 'glm' from package 'stats'.
`n.pc`	Number of principal components to use in the logistic.
`scale`	Same as in 'prcomp' from package 'prcomp'.
`center`	Same as in 'prcomp' from package 'prcomp'.
`tol`	Same as in 'prcomp' from package 'prcomp'.
`max.pc`	Same as in parameter 'rank.' from package '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', 'posterior', 'pca.ind.coord', or 'all'. If type = 'all', function 'predict.pcaLogisticR' ('predict') returns a list with: 1) 'class': individual classification. 2) 'posterior': 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, while the logistic regression is performed using function glm, both functions from R package 'stats'. The current application only use basic functionalities from the 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')
```

pcalogit_perf — *Classification logistic model for simulated dataset of DMPs used in examples*

Description

This data/object carries the information about the classification performance of the combined models of Principal Components *PCA* and logistic regression analyses on the set of `dmps`.

Usage

```
pcalogit_perf
```

Format

pcalogit_perf is an object from class *"pcaLogisticR"*, consisting of a list with the following elements:

Performance Classification performance of the *"pca.logistic"* model on the set of `dmps`.

FDR False discovery rate estimated for the logistic model on the set of `dmps`

model The fitted `glm` *"logistic"* and the *"PCA"* models. The *"PCA"* is fitted with `prcomp`function.

pcalogit_perf is an object from *"pcaLogisticR"* class, which was obtained applying `evaluateDIMPclass` function on the set of `dmps`.

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

Description

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

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

Usage

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

```
  method = "moment",
  max.pc = NULL
)

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

Arguments

formula	Same as in qda from package 'MASS'.
data	Same as in qda from package 'MASS'.
grouping	Same as in qda from package 'MASS'.
n.pc	Number of principal components to use in the qda.
scale	Same as in prcomp from package 'stats'.
center	Same as in prcomp from package 'stats'.
tol	Same as in prcomp from package 'stats'.
method	Same as in qda from package 'MASS'.
max.pc	Same as in parameter 'rank.' from prcomp from package 'stats'.
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' basic predictions: classes and posterior classification probabilities. Option 'qda.pred' returns the object given by function 'predict.qda' from MASS package: 'class', 'posterior', 'scores' (cases scores on discriminant variables, see qda.
...	Not in use.

Details

The principal components (PCs) are obtained using the function 'prcomp' from R package '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.

See Also

pcaLDA, qda and predict.lda

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 a random seed to reproduce the sampling
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)
```

---

pcaQda_perf — *Classification PCA+QDA model for simulated dataset of DMPs used in examples*

Description

This data/object carries the information about the classification performance of the combined models of Principal Components *PCA* and Quadratic Discriminant (*"QDA"*)) analyses on the set of dmps.

Usage

```
pcaQda_perf
```

Format

pcaQda_perf is an object from class *"pcaQDA"*, consisting of a list with the following elements:

Performance Classification performance of the *"PCA+QDA"* model on the set of dmps.

FDR False discovery rate estimated for the *"PCA+QDA"* model on the set of dmps.

model The *"pcaQDA"* model fitted on the set of dmps, carrying the qda and the *"PCA"* models. The *"PCA"* is fitted with prcompfunction.

pcaQda_perf is an object from "pcaQDA" class, which was obtained applying evaluateDIMPclass function on the set of dmps.

`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 = c("mean", "median", "jackmean", "sum"),
  num.cores = 1,
  tasks = 0L,
  prob = FALSE,
  column = 1L,
  jstat = c("sum", "mean", "median"),
  verbose = TRUE,
  ...
)
```

Arguments

LR	List of GRanges objects to build a virtual individual (methylation pool). It is assumed that the list of GRanges was obtained with readCounts2GRangesList. That is, the metacolumn from each GRanges object must contain the columns named 'mC' (number of reads signaling methylated cytosine) and 'uC' (number of reads signaling non-methylated cytosine). If more than two columns are carried on each GRanges object, then the parameter "columns" denoting the column numbers where "uC" and "mC" are located must be passed to uniqueGRanges function.
stat	statistic used to estimate the methylation pool: row 'mean', row 'median', row 'sum', or Jacknife row mean ('jackmean') of methylated and unmethylated read counts across individuals. Notice that, for only two samples, 'jackmean' makes not sense. Since the centrality statistics are sensitive to extreme values, stat = 'sum' is an attractive option. However, in this last case, a further correction for the minimum coverage for the reference sample must be taken into account in a furhter estimation of the Hellinger divergence of methylation levels, which is explained in the detail section from the help of function estimateDivergence. A conservative option is 'mean', which will return the group centroid.

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.
jstat	If stat = 'jackmean', then any of the 'stat' possible values: 'sum', 'mean', or 'median' can be used to compute, for each cytosine site, the Jacknife vector of the selected statistics and then to compute the corresponding mean. Default is jstat = 'sum'.
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 = 'mean', verbose = FALSE)
```

`predict.cdfMODEL` *Predict function for probability distributions in Methyl-IT*

Description

This is an utility function to get predictions from the probability distributions models used in Methyl-IT: Weibull, Gamma, and generalized Gamma. Some times, after the nonlinear fit of any of the mentioned modelsm we would like to evaluate the model output.

Usage

```
## S3 method for class 'cdfMODEL'
predict(object, pred = "quant", q = 0.95, dist.name, lower.tail = TRUE)

## S3 method for class 'cdfMODELlist'
predict(
  object,
  pred = "quant",
  q = 0.95,
  dist.name,
  num.cores = 1L,
  tasks = 0L
)
```

Arguments

`object`	An object carrying the best nonlinear fit for a distribution model obtained with function `nonlinearFitDist`.
`pred`	Type of prediction resquested: *density* ('dens'),*quantiles* ('quant'), *random number* ('rnum') or *probabilities* ('prob').
`q`	numeric vector of quantiles, probabilities or an interger if pred = 'rnum'.
`dist.name`	name of the distribution to fit: Weibull2P (default: 'Weibull2P'), Weibull three-parameters (Weibull3P), gamma with three-parameter (Gamma3P), gamma with two-parameter (Gamma2P), generalized gamma with three-parameter ('GGamma3P') or four-parameter ('GGamma4P').
`lower.tail`	logical; if TRUE (default), probabilities are $P(X \leq x)$, otherwise, $P(X > x)$)(p-value).
`num.cores, tasks`	Paramaters for parallele computation using package `BiocParallel-package`: the number of cores to use, i.e. at most how many child processes will be run simultaneously (see `bplapply` and the number of tasks per job (only for Linux OS).

Details

Predictions are based on the best model fit returned by function `nonlinearFitDist`. The possible prediction are: *density*, *quantiles*, *random number* or *probabilities*.

Examples

```
data(HD)
set.seed(13)
num.points <- 8286
HD <- HD[4]
HD$T1$hdiv <- rweibull(1:num.points, shape = 0.75, scale = 1)

nlms <- nonlinearFitDist(HD, column = 9L, verbose = FALSE)

x=seq(0.1, 10, 0.05)

## Predicted values
y <- predict(nlms[[1]], pred='prob', q = x, dist.name='Weibull2P')

## Theoretical values
y1 <- pweibull(x, shape = 0.75, scale = 1)

# The maximum difference between the 'theoretical' and estimated densities
max(abs(round(y, 2) - round(y1, 2)))
```

---

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

Description

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

Usage

```
## S3 method for class 'LogisticR'
predict(
  object,
  newdata = NULL,
  type = c("all", "class", "posterior"),
  num.cores = 1L,
  tasks = 0L,
  ...
)
```

Arguments

object	To use with function 'predict'. An object from 'LogisticR' class. A logistic model given by function evaluateDIMPclass.
newdata	To use with function 'predict'. New data for classification prediction. Optionally, an object from class 'GRanges', a list of GRanges, 'pDMP' or 'InfDIv', in which to look for variables with which to predict. If omitted, the fitted linear predictors are used.

`type` The type of output required. Possible outputs are: 'class', 'posterior' and 'all'. The default is 'all'.

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

`...` Not in use.

Details

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

Value

If type is set to 'all', then the original 'newdata' with two columns added, predicted classes and 'posterior' probabilities, in the meta-columns of each GRanges object are given. If 'newdata' is null, then the predictions given for the model by function `predict.glm` are returned. if type is set to 'class' or to 'posterior', then the unlisted predicted classification or posterior classification probabilities are returned.

---

predictDIMPclass *Predict DIMP class*

Description

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

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 methylated position (DMPs). The metacolumn of each GRanges must contain the column: Hellinger divergence 'hdiv', total variation 'TV', the probability of potential DMP '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 the 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 DMPs belong to same methylation context and derive from an experiment accomplished under the same condition set for the DMPs used to build the model.

Value

The same LR object with tow new columns named 'class' and 'posterior' added to each GRanges object from LR (default). Based on the model prediction each DMP is labeled as control 'CT' or as treatment 'TT' in column 'class'. Column 'posterior' provides, for each DMP, the posterior probability that the given DMP can be classified as induced by the 'treatment' (a treatment DMP).

```
Control DMPs classified as 'treatment' are false positives. However, if
the same cytosine position is classified as 'treatment DMP' in both
groups, control and treatment, but with higher posterior probability
in the treatment group, then this would indicate a reinforcement of the
methylation status in such a position induced by the treatment.

If 'conf.matrix' is TRUE and the arguments control.names and
treatment.names are provided, then the overall confusion matrix is
returned.
```

Examples

```
data(cutpoint, PS, package = 'MethylIT')

## DMPs are selected using the cupoints
DMPs <- selectDIMP(PS, div.col = 9L, cutpoint = cutpoint$cutpoint,
tv.cut = 0.92)

## Classification of DMPs into two clases: DMPS from control and DMPs from
## treatment samples and evaluation of the classifier performance (for more
## details see ?evaluateDIMPclass).
perf <- evaluateDIMPclass(LR = DMPs, column = c(hdiv = TRUE, TV = TRUE,
wprob = TRUE, pos = TRUE), classifier = 'lda', n.pc = 4L,
control.names =  c('C1', 'C2', 'C3'), treatment.names = c('T1', 'T2', 'T3'),
center = TRUE, scale = TRUE, prop = 0.6)
```

```
## Now predictions of DMP for control and treament can be obtained
pred = predictDIMPclass(LR = DMPs, model = perf$model, conf.matrix = TRUE,
control.names = c('C1', 'C2', 'C3'), treatment.names = c('T1', 'T2', 'T3'))
```

`PS` *Simulated dataset of potential DMPs used in examples*

Description

Each individuals sample includes 10000 cytosine positions

Usage

```
PS
```

Format

PS is an object from class 'pDMP' carrying in the meta-columns the following variables:

p1 methylation level from the reference sample

p2 methylation level from the treatment sample

TV the total variation distance (difference of methylation levels)

hdiv Hellinger divergence

wprob the probabilities: $wprob = 1 - CDFprobability$

PS is an object from class *'pDMP'* carrying the same meta-columns as 'HD' (dataset) plus the probabilities: $wprob = 1 - CDFprobability$. ***PS*** object was obtained with function `getPotentialDIMP`.

`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]) )
```

---

qda_perf *Classification LDA model for simulated dataset of DMPs used in examples*

Description

This data/object carries the information about the classification performance of a Quadratic Discriminant (*"QDA"*)) model on the set of dmps.

Usage

```
qda_perf
```

Format

qda_perf is list object consisting of the following elements:

Performance Classification performance of the *"LDA"* model on the set of dmps.

FDR False discovery rate estimated for the *"QDA"* model on the set of dmps

model The qda model fitted on the set of dmps.

qda_perf is a list object obtained applying evaluateDIMPclass function on the set of dmps.

`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, si
    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' (methylation percentage), 'mC' (methylates cytosine), 'uC' (non methylated cytosine), 'coverage', and 'context' (methylation context). These column headers are not required to be in the files. An optional column named *'signal'* can be used to include a relevant information about the methylation signal.
`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'
## '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
```

---

sortBySeqnameAndStart

Sorting 'GRanges' objects

Description

Sorts a GRanges object by seqname and start position

Usage

```
sortBySeqnameAndStart(gr)

sortBySeqnameAndEnd(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,
  keep.strand = !ignore.strand,
  num.cores = 1,
  tasks = 0L,
  verbose = TRUE
)
```

Arguments

`ListOfGranges`
: Objects to combine. A list of `GRanges-class` object or a `GRangesList-class` object.

`ncols`
: integer. Number of columns to use from the meta-column of each GRanges object. Default value: NULL. If NULL, all the columns (from column 1 to ncols) from each GRanges will be present in the $uniqueGRanges$ output.

`columns`
: integer number(s) corresponding to the specific column(s) to use from the meta-column of each GRanges. Default value: NULL. if provided, the metacolumn from the uniqueGRanges output will contain the specified columns.

`chromosomes`
: Chromosomes used Default value: NULL

`maxgap, minoverlap, ignore.strand, select, type`
: The same as in `findOverlaps-methods`.

`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 metacolumn value x is missing in the GRanges object Y, then the metacolumn of range k from uniqueGRanges(list(X,Y)) object will be the row vector (x,0) or (x,NA) if missing = NA.

`keep.strand`
: When set to TRUE, the strand information is preserved on the objects even if ignore.strand is set to TRUE. This makes it possible to ignore the strand during overlap calculations but to preserve the strand information and not overwrite with *. Default value is keep.strand = !ignore.strand.

`num.cores`
: The number of cores to use, i.e. at most how many child processes will be run simultaneously (see bplapply function from BiocParallel package).

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

`verbose`
: if TRUE, prints the function log to stdout

Details

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

Value

a GRanges object

Author(s)

Robersy Sanchez (https://genomaths.com).

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, samples '1' and '2', this function will filter by coverage each cytosine site from each GRanges object.

Usage

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

Arguments

x	An object from the classes 'GRanges', 'InfDiv', or 'pDMP' with methylated and unmethylated counts in its meta-column. If the argument 'y' is not given, then it is assumed that the first four columns of the GRanges metadata 'x' are counts: methylated and unmethylated counts for samples '1' and '2'.
y	A GRanges object with methylated and unmethylated counts in its meta-column. Default is NULL. If x is a 'InfDiv', or 'pDMP', then 'y' is not needed, since samples '1' and '2' are the first four columns of these objects.
min.coverage	An integer or an integer vector of length 2. Cytosine sites where the coverage in both samples, 'x' and 'y', are less than 'min.coverage' are discarded. The cytosine site is preserved, however, if the coverage is greater than 'min.coverage' in at least one sample. If 'min.coverage' is an integer vector, then the corresponding min coverage is applied to each sample.
min.meth	An integer or an integer vector of length 2. Cytosine sites where the numbers of read counts of methylated cytosine in both samples, '1' and '2', are less than 'min.meth' are discarded. If 'min.meth' is an integer vector, then the corresponding min number of reads is applied to each sample.
min.umeth	An integer or an integer vector of length 2. Min number of reads to consider cytosine position. Specifically cytosine positions where $uC \leq min.umethandmC > 0andmC < min.meth$ hold will be removed, where mC and uC stand for the numbers of methylated and unmethylated reads. Default is $min.umeth = 0$.
min.sitecov	An integer. The minimum total coverage. Only sites where the total coverage $cov1 + cov2$ is greater than 'min.sitecov' are considered for downstream analysis, where cov1 and cov2 are the coverages for samples 1 and 2, respectively.
percentile	Threshold to remove the outliers (PCR bias) from each file and all files stacked. If 'high.coverage = NULL', then the threshold q will be computed as: $$q1 = quantile(cov1, probs = percentile)$$ $$q2 = quantile(cov2, probs = percentile)$$ $$q = min(q1, q2)$$ where \eqn{cov1} and \eqn{cov2} are the coverage vectors from samples and 2, respectively.
high.coverage	An integer for read counts. Cytosine sites having higher coverage than this are discarded. Default is NULL. If **high.coverage** is not NULL, then the **percentile** argument is disregarded and **high.coverage** is used as threshold to remove the PCR bias. If percentile is not null, then q = max(q, high.coverage).
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).

`ignore.strand`
: When set to TRUE, the strand information is ignored in the overlap calculations. Default value: TRUE

`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' in at least one of the samples are preserved. Positions with 'coverage' < 'min.coverage' in both samples, 'x' and 'y', are removed. Positions with 'coverage' > 'percentile' (e.g., 99.9 percentile) are removed as well. 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

```
### Create new data
df1 <- data.frame(chr = 'chr1', start = 11:16, end = 11:16,
                  mC = c(2,10,7,9,1,10), uC = c(30,20,4,8,0,10))

df2 <- data.frame(chr = 'chr1', start = 12:18, end = 12:18,
                  mC2 = 1:7, uC2 = 0:6)

gr1 <- makeGRangesFromDataFrame(df1, keep.extra.columns = TRUE)
gr2 <- makeGRangesFromDataFrame(df2, keep.extra.columns = TRUE)

## Filtering
r1 <- uniqueGRfilterByCov(gr1, gr2, ignore.strand = TRUE)
r1

## Cytosine position with coordinate 15 (rows #2) can pass the
## filtering conditions of min.coverage = 4 and lead to meaningless
## situations with methylation levels p = 1/(1 + 0) = 1
r1[2]

## The last situation can be prevent, in this case, by setting
## min.meth = 1:
r1 <- uniqueGRfilterByCov(gr1, gr2, min.meth = 1, ignore.strand = TRUE)
r1
```

`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,
  model = c("all", "2P", "3P", "Weibull2P", "Weibull3P"),
  npoints = NULL,
  maxiter = 1024,
  tol = 1e-12,
  ftol = 1e-12,
  ptol = 1e-12,
  minFactor = 10^-6,
  nlms = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

`X`	numerical vector
`sample.size`	size of the sample
`model`	Distribution model to fit, two-parameters and three-parameters Weibull model ('Weibull2P' or simply '2P' and 'Weibull3P' or '3P). Default is 'all' and the model with the best AIC criterion is reported.
`npoints`	number of points used in the fit
`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
nlms	Logical. Whether to return the nonlinear model object nls.lm. Default is FALSE.
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 covariance matrix.

Author(s)

Robersy Sanchez - 06/03/2016 https://github.com/genomaths

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