# Package ‘MethylIT.utils’

April 20, 2019

**Title** MethylIT utility

**Version** 0.1

**Encoding** UTF-8

**Author** Robersy Sanchez

**Maintainer** Robersy Sanchez <rus547@psu.edu>

**Description** MethylIT_utils is an R package with functions to support MethylIT downstream analysis.

**Depends** R (>= 3.4.0),

**License** file LICENSE

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

**LazyData** yes

**Imports** BiocParallel,
GenomicRanges,
grDevices,
genefilter,
IRanges,
stats,
betareg,
utils,
mclust,
mixtools,
caret,
S4Vectors,
stats,
MASS

**RoxygenNote** 6.1.0

**Suggests** testthat

## R topics documented:

---

`.colorBar` *Color bar for heatmaps*

---

## Description

This function is for internal use. It creates a color scaled bar for heatmap.

## Usage

```
.colorBar(z, zlim, col = heat.colors(12), breaks, horiz = TRUE,
  ylim = NULL, xlim = NULL, lwd = 0.5, cex.axis = 1, ...)
```

## Arguments

| | |
|---|---|
| `z` | matrix of values used to visualize the heatmap |
| `zlim` | Limit for the numerical(color) scale, which must be consistent with parameter ’break’. |
| `col` | Palette of color to use |
| `breaks` | Numerical vector with the breaks used to produce the heatmap |
| `horiz` | Whether to the color bar will be horizontal(= TRUE) or vertical(= FALSE) |
| `ylim` | User defined limits for y-axis. Depending on the orientation, x- or y-limits may be defined that are different from the z-limits and will reduce the range of colors displayed. |

`xlim` User defined limits for x-axis

`lwd` Line width.

`cex.axis` Cex values for color bar axis labels.

`...` Additional parameter to pass to 'par' R function

### Value

Image with color bar

## `.evaluateModel` *Evaluate a model using the Akaike information criterion (AIC)*

### Description

Evaluate a glm object using the Akaike information criterion (AIC)

### Usage

```
.evaluateModel(model, test = c("Wald", "LRT"))
```

### Arguments

`model` glm object

`test` A character string matching one of "Wald" or "LRT". If test = "Wald", then the p-value of the Wald test for the coefficient of the independent variable (*treatment group*) will be reported p-value. If test = "LRT", then the p-value from a likelihood ratio test given by `anova` function from *stats* packages will be the reported p-value for the group comparison.

### Value

AIC value

## `bootstrap2x2` *bootstrap2x2*

### Description

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

### Usage

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

## Arguments

`x` A numerical matrix corresponding to cross tabulation (2x2) table (contingency table).

`stat` Statistic to be used in the testing: ’rmst’,’hdiv’, or "all".

`num.permut` Number of permutations.

## Details

For goodness-of-fit the following null hypothesis is tested $H_\theta : p = p(\theta)$ To conduct a single simulation, we perform the following three-step procedure [1,2]:

1. To generate m i.i.d. draws according to the model distribution $p(\theta)$, where $\theta'$ is the estimate calculated from the experimental data,
2. To estimate the parameter $\theta$ from the data generated in Step 1, obtaining a new estimate $\theta$est.
3. To calculate the statistic under consideration (HD, RMST), using the data generated in Step 1 and taking the model distribution to be $\theta$est, where $\theta$est is the estimate calculated in Step 2 from the data generated in Step 1.

After conducting many such simulations, the confidence level for rejecting the null hypothesis is the fraction of the statistics calculated in step 3 that are less than the statistic calculated from the empirical data. The significance level $\alpha$ is the same as a confidence level of $1 - \alpha$.

## Value

A p-value probability

## References

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

## Examples

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

## classPerform *Classification performance based on divergences of methylation levels*

### Description

The classification performance based on an information divergence (e.g., Hellinger divergence) carried in a list of GRanges objects. The total variation distance (TVD, absolute difference of methylation levels) is used as pivot to specify the cytosine sites considered as true positives and true negatives. Function `confusionMatrix` from package "caret" is applied to get the classification performance.

### Usage

```
classPerform(LR, min.tv = 0.25, tv.cut, cutoff, tv.col, div.col = NULL,
  pval.col = NULL, stat = 1)
```

### Arguments

`LR` A list of GRanges, a GRangesList, a CompressedGRangesList object. Each GRanges object from the list must have two columns: methylated (mC) and unmethylated (uC) counts. The name of each element from the list must coincide with a control or a treatment name.

`min.tv` Minimum value for the total variation distance (TVD; absolute value of methylation levels differences, $TVD = abs(TV)$). Only sites/ranges k with $TVD_k > min.tv$ are analyzed. Defaul min.tv = 0.25.

`tv.cut` A cutoff for the total variation distance to be applied to each site/range. If tv.cut is provided, then sites/ranges k with $TVD_k < tv.cut$ are considered TRUE negatives and $TVD_k > tv.cut$ TRUE postives. Its value must be NULLor a number $0 < tv.cut < 1$.

`cutoff` A divergence of methylation levels or a p-value cutoff-value for the the magnitude given in div.col or in pval.col, respectively (see below). The values greater than 'cutoff' are predicted TRUE (positives), otherwise are predicted FALSE (negatives).

`tv.col` Column number for the total variation distance (TVD; absolute value of methylation levels differences, $TVD = abs(TV)$).

`div.col` Column number for divergence variable used in the performance analysis and estimation of the cutpoints. Default: NULL. One of the parameter values div.col or pval.col must be given.

`pval.col` Column number for p-value used in the performance analysis and estimation of the cutpoints. Default: NULL. One of the parameter values div.col or pval.col must be given.

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

## 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 unmathylated read counts for each group (control and treatment) in the metacolumn. So, a contingence table can be built for range from GRanges object.

## Value

A list with the classification repformance results

## Author(s)

Robersy Sanchez

## Examples

```
# load simulated data of potential methylated signal
data(sim_ps)

classPerform(LR = PS, min.tv = 0.25, tv.cut = 0.4,
             cutoff = 68.7, tv.col = 7L, div.col = 9, stat = 0)
```

# divTest *Group Comparisons of Information Divergences Based on Generalized Linear Model*

## Description

Generalized Linear Model for group comparison of information divergence variables yielded by MethylIT output. Basically, this a wrapping function to perform the fitting of generalized linear models with glm from 'stats' package to any variable of interest given in GRanges objects of MethylIT output.

## Usage

```
divTest(GR, control.names, treatment.names, glm.family = Gamma(link =
  "log"), var.weights = FALSE, weights = NULL, varFilter = 0,
  meanFilter = 0, FilterLog2FC = TRUE, Minlog2FC = 1,
  divPerBp = 0.001, minInd = 2, pAdjustMethod = NULL, scaling = 1L,
  pvalCutOff = 0.05, saveAll = FALSE, num.cores = 1, tasks = 0L,
  verbose = TRUE, ...)
```

## Arguments

GR GRanges objects including control and treatment samples containing an information divergence of methylation levels. The names for each column must coincide with the names given for parameters: 'control.names' and 'treatment.names'.

`control.names` Names/IDs of the control samples, which must be included in the variable GR in a metacolumn.

`treatment.names` Names/IDs of the treatment samples, which must be included in the variable GR in a metacolumn.

`glm.family, link` Parameter to be passed to function `glm`. A description of the error distribution and link function to be used in the model. For `glm` this can be a character string naming a family function, or the result of a call to a family function. For `glm`.fit only the third option is supported. (See`family` function). Default: glm.family=Gamma(link ="log").

`var.weights` Logical (default: FALSE). Whether to use group variances as weights.

`weights` An optional list of two numeric vectors of ‘prior weights’ to be used in the fitting process. One vector of weights for the control and one for the treatment. Each vector with length equal to length(GR) (default: NULL). Non-NULL weights can be used to indicate that different observations have different dispersions (with the values in weights being inversely proportional to the dispersions).

`varFilter` Numeric (default: 0). GLM will be performed only for those rows (ranges denoting genomic regions) where the group variance is greater the number specified by varFilter.

`meanFilter` Numeric (default: 0). GLM will be performed only for those rows (ranges denoting genomic regions) where the absolute difference of group means is greater the number specified by meanFilter.

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

`Minlog2FC` minimum logarithm base 2 of fold changes

`divPerBp` At least for one group the mean divergence per bp must be equal to or greater than ’divPerBp’ (default divPerBp = 0.001).

`minInd` Integer (Default: 2). At least one group must have ’minInd’ individuals with a divergence value greater than zero.

`pAdjustMethod` Method used to adjust the results; default: "NULL" (see `p.adjust`.methods). The p-value adjustment is performed using function `p.adjust`.

`scaling` integer (default 1). Scaling factor estimate the signal density as: scaling * "DIMP-Count-Per-Bp". For example, if scaling = 1000, then signal density denotes the number of DIMPs in 1000 bp.

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

`saveAll` if TRUE all the temporal results that passed filters ’varFilter’ and are ’meanFilter’ returned. If FALSE, only the comparisons that passed filters ’varFilter’, ’meanFilter’, and pvalue < pvalCutOff or adj.pvalue < pvalCutOff (if pAdjustMethod is not NULL) are returned.

`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).

`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-class` from BiocParallel package).

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

`...` Additional parameters passed to `glm` function.

## Details

The default parameter setting glm.family = Gamma(link = "log") is thought to perform the group comparison of the sums of absolute differences of methylation levels (total variation distance (TVD) at gene-body DIMPs on DMGs). The sums of Hellinger divergence (HD, at gene-body DIMPs on DMGs) can be tested with this setting as well. Both TVD and HD follow asymptotic Chi-square distribution and, consequently, so do the sum of TVD and the sum of HD. The Chi-square distribution is a particular case of Gamma distribution:

$$f(x|a,s) = 1/(s^aGamma(a))x^(a-1)e^-(x/s)$$

Chi-square density is derived after replacing a = n/2 and s = 2:

$$f(x|n) = 1/(2^(n/2)Gamma(n/2))x^(n/2-1)e^( - x/2)$$

## Value

The original GRanges object with the columns "beta", "log2FC", "pvalue", "adj.pval" (if pAdjustMethod requested), "CT.divPerBp" and "TT.divPerBp" (divergence per base pairs), and "divPerBpVariation added.

## 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"))
# === The number of cytosine sites to generate ===
sites = 11001
# == Set a seed for pseudo-random number generation ===
set.seed(123)
alpha.ct <- 0.09
alpha.tt <- 0.2
# === Simulate samples ===
ref = simulateCounts(num.samples = 2, sites = sites, alpha = alpha.ct,
                   beta = 0.5, size = 50, theta = 4.5, sample.ids = "R1")

# Control group
ctrl = simulateCounts(num.samples = 2, sites = sites, alpha = alpha.ct,
                       beta = 0.5, size = 50, theta = 4.5,
                      sample.ids = c("C1", "C2"))
# Treatment group
treat = simulateCounts(num.samples = 2, sites = sites, alpha = alpha.tt,
                       beta = 0.5, size = 50, theta = 4.5,
                       sample.ids = c("T1", "T2"))

#  === Estime Divergences ===
HD = estimateDivergence(ref = ref$R1, indiv = c(ctrl, treat),
                       Bayesian = TRUE, num.cores = 1L, percentile = 1,
                       verbose = FALSE)

nlms <- nonlinearFitDist(HD, column = 4, verbose = FALSE)

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

## The cutpoint estimation used to discriminate the signal from the noise
cutpoints <- estimateCutPoint(PS, control.names = c("C1", "C2"),
                             treatment.names = c("T1", "T2"),
                             div.col = 4, verbose = FALSE)
## DIMPs are selected using the cupoints
DIMPs <- selectDIMP(PS, div.col = 9, cutpoint = min(cutpoints$cutpoint))

## Finally DIMPs statistics genes
tv_DIMPs = getGRegionsStat(GR = DIMPs, grfeatures = genes, stat = "sum",
                          absolute = TRUE, column = 7L)

GR_tv_DIMP = uniqueGRanges(tv_DIMPs, type = "equal", chromosomes = "1")
colnames(mcols(GR_tv_DIMP)) <-  c("C1", "C2", "T1", "T2")

res <- divTest(GR=GR_tv_DIMP, control.names =  c("C1", "C2"),
              treatment.names = c("T1", "T2"))
```

## dmpDensity *Linear density of DMPs at a given genomic region*

### Description

The linear density of DMPs in a given genomic region (GR) is defined according with the classical terminology in physics, i.e., as the measure of the physical quantity of any characteristic value per unit of length. In the current case, as the amount of DIMPs per nucleotide base.

### Usage

```
dmpDensity(GR, column = 1, cut.col = 1, cutoff, Chr = NULL,
  start.pos = NULL, end.pos = NULL, int.size1 = NULL,
  int.size2 = NULL, breaks = NULL, scaling = TRUE, plot = FALSE,
  noDMP.dens = TRUE, xlabel = "Coordinate",
  ylabel = "Normalized density", col.dmp = "red", col.ndmp = "blue",
  yintercept = 0.25, col.yintercept = "magenta",
  type.yintercept = "dashed", dig.lab = 3)
```

## Arguments

`GR` A genomic GRanges object carrying the genomic region where the estimation of the DMP density will be accomplished.

`cut.col` Integer denoting the GR metacolumn where the decision variable about whether a position is DMP is located. Default cut.col = 1.

`cutoff` Cut value to decide wheter the value of the variable used to estimate the density is a DMP at each position. If missing, then cutoff is estimated as the first queantile greater than zero from the values given in the GR column *cut.col*.

`Chr` A character string. Default NULL. If the GR object comprises several chromosomes, then one chromosome must be specified. Otherwise the density of first chromosome will be returned.

`start.pos, end.pos`
Start and end positions, respectively, of the GR where the density of DMPs will be estimated. Default NULL. If NULL densities will be estimated for the whole GR and the specified chromosome.

`int.size1, int.size2`
The interval/window size where the density of DMP and no DMPs are computed. Default Null.

`breaks` Integer. Number of windows/intervals to split the GR. Deafult NULL. If provided, then it is applied to compute the densities of DMPs and no-DMPs. If 'int.size1', 'int.size2', and 'breaks' are NULL, then the breaks are computed as: `breaks <- min(150, max(start(x))/nclass.FD(start(x)),na.rm = TRUE)`, where function *nclass.FD* (`nclass`) applies Freedman-Diaconis algorithm.

`scaling` Logic value to deside whether to perform the scaling of the estimated density values or not. Default is TRUE.

`plot` Logic. Whether to produce a grahic or not. Default, plot = TRUE.

`noDMP.dens` Logic whether to produce the graphics for no-DMP density. Default is TRUE

`xlabel` X-axis label. Default *xlabel = "Coordinate"*.

`ylabel` Y-axis label. Default *ylabel = "Normalized density"*.

`col.dmp` Color for the density of DMPs in the graphic.

`col.ndmp` Color for the density of no DMPs in the graphic.

`yintercept` If plot == TRUE, this is the position for an horizantal line that intercept the y-axis. Default yintercept = 0.25.

`col.yintercept` Color for the horizantal line 'yintercept'. Default *col.yintercept = 'blue'*

`type.yintercept`
Line type for the horizantal line 'yintercept'. Default *type.yintercept = "dashed"*.

`dig.lab` integer which is used when labels are not given. It determines the number of digits used in formatting the break numbers.

## Details

Since the number of DIMPs along the DNA sequence vary, the local density of DMPs $\rho_i$ at a fixed interval Δ l_i is defined by the quotient $\rho_i = \Delta DMP_i/\Delta l_i$ is the amount of DIMPs at the fixed interval. Likewise the local density of non-DIMPs is defined as $\rho_i = \Delta nonDMP_i/\Delta l_i$. Notice that for a specified methylation context, e.g., CG, $\Delta CG_i - \Delta DMP_i$, where $\Delta CG$ is the amount CG positions at the given interval. The linear densities are normalized as $\rho_i/\rho_m ax$, where $\rho_m ax$ is the maximum of linear density found in a given GR.

## Value

If plot is TRUE will return a graphic with the densities of DMPs and and no DMPs. If plot is FALSE a data frame object with the density of DMPs and not DMPs will be returned.

## Author(s)

Robersy Sanchez

## Examples

```
set.seed(349)
## An auxiliary function to generate simulated hypothetical values from a
## variable with normal distribution
hypDT <- function(mean, sd, n, num.pos, noise) {
    h <- hist(rnorm(n, mean = mean, sd = sd), breaks = num.pos, plot = FALSE)
    hyp <- h$density * 60 + runif(length(h$density)) * noise
    return(hyp)
}

## To generate a matrix of values with variations introduced by noise
hyp <- hypDT(mean = 5, sd = 30, n = 10^5, noise = 4, num.pos = 8000)
## A GRanges object is built, which will carries the previous matrix on its
## meta-columns
l <- length(hyp)
starts <- seq(0, 30000, 3)[1:l]
ends <- starts
GR <- GRanges(seqnames = "chr1", ranges = IRanges(start = starts,
                end = ends))
mcols(GR) <- data.frame(signal = hyp)

# If plot is TRUE a grphic is printed. Otherwise data frame is returned.
p <- dmpDensity(GR, plot = FALSE)

# If ggplot2 package is installed, then graphic can customized using
# the returned data frame 'p':

# library(ggplot2)
## Auxiliar function to write scientific notation in the graphics
# fancy_scientific <- function(l) {
#   #'turn in to character string in scientific notation
#   l <- format( l, scientific = TRUE, digits = 1 )
#   l <- gsub("0e\\+00","0",l)
#   #'quote the part before the exponent to keep all the digits
#   l <- gsub("^(.*)e", "'\\1'e", l)
#   #'turn the 'e+' into plotmath format
#   l <- gsub("e", "%*%10^", l)
#   l <- gsub("[+]", "", l )
#   #'return this as an expression
#   parse(text=l)
# }
#
# max.pos = max(p$DMP.coordinate)
# ggplot(data=p) +
#   geom_line(aes(x=DMP.coordinate, y=DMP.density), color="red") +
#   geom_hline(aes(yintercept=0.25), linetype="dashed",
#              colour="blue", show.legend=FALSE ) +
#   geom_line(aes(x=coordinate, y=density), color="blue") +
#   xlab("Coordinate") + ylab("Normalized density") +
#   scale_y_continuous(breaks=c(0.00, 0.25, 0.50, 0.75, 1.00)) +
#   scale_x_continuous(breaks=c(0.00, 0.25 *max.pos, 0.50*max.pos,
#                               0.75*max.pos, max.pos),
#                      labels = fancy_scientific) +
#   expand_limits(y=0)
```

## evalDetection *Evaluate detection performance of a signal detector*

### Description

For a given cutpoint (e.g., previously estimated with the function estimateCutPoint), 'evalDetection' will return the evaluation of the methylation signal into two clases: signal from control and signal from treatment samples.

### Usage

```
evalDetection(LR, control.names, treatment.names, cutpoint, div.col = 7L,
  seed = 1234, verbose = TRUE)
```

### Arguments

`LR` A list of GRanges objects (LR) including control and treatment GRanges containing divergence values for each cytosine site in the meta-column. LR can be generated, for example, by the function `estimateDivergence`. 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.

`cutpoint` Cutpoint to select DIMPs. Cytosine positions with divergence greater than 'cutpoint' will selected as DIMPs. Cutpoints are estimated with the function 'estimateCutPoint'. Cytosine positions with divergence values greater than the cutpoint are considered members of the "positive class".

`div.col` Column number for divergence variable used in the ROC analysis and estimation of the cutpoints.

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 under natural, ordinary conditions. Here, to evaluate the performance of signal classification obtained with the application of some classifier/detector or rule, the cross-tabulation of observed and predicted classes with associated statistics are calculated with function `confusionMatrix` fron package "caret".

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

the list with the statisitics returned by the function `confusionMatrix` fron package "caret".

## Examples

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

#'#' List of Granges objects of individuals methylation levels
Indiv <- GRangesList(
  sample11 = makeGRangesFromDataFrame(
    data.frame(chr = '1',
               start = 1:num.points,
               end = 1:num.points,
               strand = '*',
               p2 = rbeta(num.points, shape1 = 1.5, shape2 = 2)),
    keep.extra.columns = TRUE),
  sample12 = makeGRangesFromDataFrame(
    data.frame(chr = '1',
               start = 1:num.points,
               end = 1:num.points,
               strand = '*',
               p2 = rbeta(num.points, shape1 = 1.6, shape2 = 2)),
    keep.extra.columns = TRUE),
  sample21 = makeGRangesFromDataFrame(
    data.frame(chr = '1',
               start = 1:num.points,
               end = 1:num.points,
               strand = '*',
               p2 = rbeta(num.points, shape1 = 40, shape2 = 4)),
    keep.extra.columns = TRUE),
  sample22 = makeGRangesFromDataFrame(
    data.frame(chr = '1',
               start = 1:num.points,
               end = 1:num.points,
               strand = '*',
               p2 = rbeta(num.points, shape1 = 41, shape2 = 4)),
    keep.extra.columns = TRUE))
#'#' To estimate Hellinger divergence using only the methylation levels.
HD <- estimateDivergence(ref = Ref, indiv = Indiv, meth.level = TRUE,
                         columns = 1)
res <- evalDetection(LR  = HD, control.names = c("sample11", "sample12"),
                     treatment.names = c("sample21", "sample22"),
                     cutpoint = 0.85, div.col = 3L, seed=1234, verbose=TRUE)
```

## findCutpoint *Find a cutoff of divergences of methylation level values*

### Description

A function to help on the decision of which is the best cutoff value for DIMP/DMP predictions. The genome-wide methylation changes that occurs in any living organism is the result of the superposition of several stochastic processes: the inherent stochasticity of biological processes and, particular, ultimately, it derives from the stochasticity of biochemical processes. On this scenario, there is not way to say with absolute determinism where a given value of an information divergence is a true positive value or a true negative value. All what we can do is the estimation of performance indicators like accuracy, sensitivity, false positive rate, etc., to evaluate the consequences of our decision on what we consider a true positive or a true negative. For example, a difference of methylation levels of 100 samples in given cytosine position does not means that this difference will not be observed in some sample from the control group. Without any doubt about it, such a different can be found in control samples as well. The fluctuation theorem guaranty such an outcome, which in the current context is a consequence of the action of second law of thermodynamics on living organisms.

### Usage

```
findCutpoint(LR, min.tv = 0.25, tv.cut = 0.5, predcuts, tv.col,
  div.col = NULL, pval.col = NULL, stat = 1, maximize = TRUE,
  num.cores = 1L, tasks = tasks)
```

### Arguments

`LR` A list of GRanges, a GRangesList, a CompressedGRangesList object. Each GRanges object from the list must have at least two columns: a column containing the total variation of methylation level (TV, difference of methylation levels) and a column containing a divergence of methylation levels (it could be TV or Hellinger divergence) or a column with a p-value from where the cutpoint will be found (see example).

`min.tv` Minimum value for the total variation distance (TVD; absolute value of methylation levels differences, $TVD = abs(TV)$). Only sites/ranges k with $TVD_k > min.tv$ are analyzed. Defaul min.tv = 0.25.

`tv.cut` A cutoff for the total variation distance to be applied to each site/range. Sites/ranges k with $TVD_k < tv.cut$ are considered TRUE negatives and sites with $TVD_k > tv.cut$ TRUE positives. Its value must be a number $0 < tv.cut < 1$. A possible value for tv.cut would be, e.g., the minimum value of *TV* found in the treatment group after the potential DMPs are estimated. Default is $tv.cut = 0.5$.

`predcuts` A numerical vector of possible cutoff values (cutpoints) for a divergence of methylation levels value or a p-value, according with the magnitude given in div.col or in pval.col, respectively. For each cutpoint k the values greater than predcuts[k] are predicted TRUE (positives), otherwise are predicted FALSE (negatives).

`tv.col` Column number where the total variation is located in the metadata from each GRanges object.

`div.col` Column number for divergence of methylation levels used in the estimation of the cutpoints. Default: NULL. One of the parameter values div.col or pval.col must be given.

`pval.col` Column number for p-value used in the estimation of the cutpoints. Default: NULL. One of the parameter values div.col or pval.col must be given.

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

`maximize` Whether to maximize the performance indicator given in parameter 'stat'. Default: TRUE.

`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

Given a numerical vector of cutoff values for the divergences of methylation level values, or p-values cutoffs, this function search for the cutoff value that yield the best classification performance for the specified performance indicator.

## Value

A list with the classification repformance results for the best cutoff value in the ranges of predcuts supplied.

### Author(s)

Robersy Sanchez

### Examples

```
# load simulated data of potential methylated signal
data(sim_ps)

# Vector of cutoff values
cuts = c(2, 5, 10, 15, 18, 20, 21, 22, 25, 27, 30, 35, 40,
       45, 50, 55, 60)
# === To find the cutpoint that maximize the accuracy ===
pre.cut.acc = findCutpoint(LR = PS, min.tv = 0.25, tv.cut = 0.5,
                           predcuts = cuts, tv.col = 7L, div.col = 9,
                           stat = 1, num.cores = 15)
```

## gammaMixtCut *Cutpoint estimation based on Mixtures of Gamma Distributions*

### Description

This functions estimates cutpoint value to classify DMPs into two classes: 1) from treatment and 2) from control, based on Mixtures of Gamma Distributions. The cutpoint estimations are limited to the analysis of mixture distributions of the form: $F(x) = \lambda G(x) + (1 - \lambda) H(x)$, where $\lambda \in [0, 1]$, $G(x)$ and $F(x)$ are the gamma cummulative distribution functions distributions followed by the information divergences estimated for individuals from control and treatment populations, respectively.

### Usage

```
gammaMixtCut(LR, post.cut = 0.5, div.col = NULL, tv.col = NULL,
  tv.cut = 0.25, find.cut = FALSE, control.names = NULL,
  treatment.names = NULL, column = c(hdiv = FALSE, TV = FALSE, wprob =
  FALSE, pos = FALSE), classifier = c("logistic", "pca.logistic", "lda",
  "svm", "qda", "pca.lda", "pca.qda"), prop = 0.6, clas.perf = FALSE,
  cut.interval = c(0.5, 0.8), cut.incr = 0.01, stat = 1,
  maximize = TRUE, num.cores = 1L, tasks = 0L,
  tol = .Machine$double.eps^0.5, maxiter = 1000, ...)
```

### Arguments

LR — A "pDMP"or "InfDiv" object obtained with functions getPotentialDIMP or estimateDivergence. These are list of GRanges objects, where each GRanges object from the list must have at least two columns: a column containing the total variation of methylation level (TV, difference of methylation levels) and a column containing a divergence of methylation levels (it could be TV or Hellinger divergence).

`post.cut` Posterior probability to dicide whether a DMPs belong to treatment group. Default *post.cut* = 0.5.

`div.col` Column number for divergence of methylation levels used in the estimation of the cutpoints. Default: 9L (hdiv column from an InfDiv object).

`tv.col` Column number where the total variation is located in the metadata from each GRanges object.

`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 $tv.cut = 0.25$.

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

`control.names, treatment.names`

Optional. Names/IDs of the control and treatment samples, which must be include in the variable LR (default, NULL). However, these are required if any of the parameters *find.cut* or *clas.perf* are set TRUE.

`treatment.names`

Optional. Names/IDs of the treatment samples, which must be include in the variable LR (default, NULL).

`column` a logical vector for column names for the predictor variables to be used: Hellinger divergence "hdiv", total variation "TV", probability of potential DIMP "wprob", and the relative cytosine site position "pos" in respect to the chromosome where it is located. The relative position is estimated as (x - x.min)/(x.max - x), where x.min and x.max are the maximum and minimum for the corresponding chromosome, repectively. If "wprob = TRUE", then Logarithm base-10 of "wprob" will be used as predictor in place of "wprob" ( see `evaluateDIMPclass`).

`classifier` Classification model to use. Option "logistic" applies a logistic regression model; option "lda" applies a Linear Discriminant Analysis (LDA); "qda" applies a Quadratic Discriminant Analysis (QDA), "pca.logistic" applies logistic regression model using the Principal Component (PCs) estimated with Principal Component Analysis (PCA) as predictor variables. pca.lda" applies LDA using PCs as predictor variables, and the option "pca.qda" applies a Quadratic Discriminant Analysis (QDA) using PCs as predictor variables. 'SVM' applies Support Vector Machines classifier from R package e1071.

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

`clas.perf` Logic. Whether to return the classificaiton performance for the estimated cutpoint. Default, FALSE.

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

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

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

`maximize` Whether to maximize the performance indicator given in parameter 'stat'. Default: TRUE.

`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).

`...` Additional arameters to pass to functions `evaluateDIMPclass` and `gammamixEM`.

## Details

After the estimation of potential DMPs, the pool of DMPs from control and treatment is assumed that follows mixtures of Gamma distributions corresponding to two populations. A posterior probability 2d-vector is estimated for each DMP. The cutpoint is determined from the intersection of the two gamma probabilities density distributions. That is, $f(x) and g(x)$ are the estimatied densities for control and treatment groups, repectively, then the cutpoint is the values of x for which $f(x) = g(x)$.

The Mixtures of Gamma Distributions (MGD) is estimated by using function `gammamixEM` from package *mixtools*. By default function `gammamixEM` produces returns a long list including the posterior probability to belong to the treatment. Here, by the sake of brevety only the information on the fitted model is given. The posterior model probability can be retrieved by using *predict* function. Accordign with MGD model, DMPs with a posterior probability to belong to the treatment group greater than *post.cut = 0.5* is classified as *DMP from treatment*. The post.cut can be modified. For all the cases $0 < post.cut < 1$. The cutpoint and, hence, the classification derived throught MGD model might differ from that provided throught `evaluateDIMPclass` function, which includes more information about the DMP and, therefore, reports better performance. The classification perfomance reported when *clas.perf* = TRUE or *find.cut* = TRUE is created with function `evaluateDIMPclass` for the especified matchin learning model.

If parameter *find.cut = TRUE*, then a search for the best cutpoint in a predifined inteval (*cut.interval*) is performed calling function `evaluateDIMPclass`.

---

`GeneUpDownStream` *Get Genes plus Up and Down Stream Regions*

---

## Description

Given a genes region or genomic region (GR), this function yields the GR plus the especified amount of DNA bases upstream and downstream the GR.

## Usage

```
GeneUpDownStream(GR, upstream = 0, downstream = 0, onlyUP = FALSE,
  onlyDown = FALSE)
```

### Arguments

| | |
|---|---|
| GR | A GRanges-class object containing the ranges of the genes or genomic regions to be extended upstream/downstream |
| upstream | Integer (Default: 0). The amount of DNA bases (bps) upstream of the GR. |
| downstream | Integer (Default: 0). The amount of DNA bases (bps) downstream of the GR. |
| onlyUP | Logic (Default: FALSE). If TRUE returns the region upstream the GR. |
| onlyDown | Logic (Default: FALSE). If TRUE returns the region downstream the GR. |

### Details

Users can select whether to request only upstream, only downstream, or both, upstream and downstream.

### Examples

```
starts = c(65419, 450703, 923928, 944204)
ends = c(71585, 451697, 944581, 959309)
chrs = c(rep("chr1", 2), rep("chr2", 2))
gr = makeGRangesFromDataFrame(
        data.frame(seqnames = chrs, start = starts, end = ends,
                strand = c("+", "-", "+", "-"),
                genes = c("A", "B", "C", "D")), keep.extra.columns = TRUE)

gr1 = GeneUpDownStream(GR = gr, upstream = 2000, downstream = 1000)
```

## getGRegionsStat-methods

*Statistic of Genomic Regions*

### Description

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

### Usage

```
getGRegionsStat(GR, win.size = 350, step.size = 350,
  grfeatures = NULL, stat = c("sum", "mean", "gmaean", "median",
  "density"), absolute = FALSE, select.strand = NULL, column = 1L,
  prob = FALSE, entropy = FALSE, maxgap = -1L, minoverlap = 0L,
  scaling = 1000L, logbase = 2, type = c("any", "start", "end",
  "within", "equal"), ignore.strand = FALSE, na.rm = TRUE,
  num.cores = 1L, tasks = 0, ...)

## S4 method for signature 'GRanges'
getGRegionsStat(GR, win.size, step.size, grfeatures,
  stat, absolute, select.strand, column, prob, entropy, maxgap, minoverlap,
  scaling, logbase, type, ignore.strand, na.rm)

## S4 method for signature 'list'
getGRegionsStat(GR, win.size = 350, step.size = 350,
  grfeatures = NULL, stat = c("sum", "mean", "gmaean", "median",
  "density"), absolute = FALSE, select.strand = NULL, column = 1L,
  prob = FALSE, entropy = FALSE, maxgap = -1L, minoverlap = 0L,
  scaling = 1000L, logbase = 2, type = c("any", "start", "end",
  "within", "equal"), ignore.strand = FALSE, na.rm = TRUE,
  num.cores = 1L, tasks = 0, ...)

## S4 method for signature 'InfDiv'
getGRegionsStat(GR, win.size = 350, step.size = 350,
  grfeatures = NULL, stat = c("sum", "mean", "gmaean", "median",
  "density"), absolute = FALSE, select.strand = NULL, column = 1L,
  prob = FALSE, entropy = FALSE, maxgap = -1L, minoverlap = 0L,
  scaling = 1000L, logbase = 2, type = c("any", "start", "end",
  "within", "equal"), ignore.strand = FALSE, na.rm = TRUE,
  num.cores = 1L, tasks = 0, ...)

## S4 method for signature 'pDMP'
getGRegionsStat(GR, win.size = 350, step.size = 350,
  grfeatures = NULL, stat = c("sum", "mean", "gmaean", "median",
  "density"), absolute = FALSE, select.strand = NULL, column = 1L,
  prob = FALSE, entropy = FALSE, maxgap = -1L, minoverlap = 0L,
  scaling = 1000L, logbase = 2, type = c("any", "start", "end",
  "within", "equal"), ignore.strand = FALSE, na.rm = TRUE,
  num.cores = 1L, tasks = 0, ...)

## S4 method for signature 'GRangesList'
getGRegionsStat(GR, win.size = 350,
  step.size = 350, grfeatures = NULL, stat = c("sum", "mean",
  "gmaean", "median", "density"), absolute = FALSE,
  select.strand = NULL, column = 1L, prob = FALSE, entropy = FALSE,
  maxgap = -1L, minoverlap = 0L, scaling = 1000L, logbase = 2,
  type = c("any", "start", "end", "within", "equal"),
  ignore.strand = FALSE, na.rm = TRUE, num.cores = 1L, tasks = 0,
  ...)
```

## Arguments

`GR` A Grange object with the variable of interest in its metacolumn.

`win.size` An integer for the size of the windows/regions size of the intervals of genomics regions.

`step.size` Interval at which the regions/windows must be defined

`grfeatures` A GRanges object corresponding to an annotated genomic feature. For example, gene region, transposable elements, exons, intergenic region, etc. If provided, then parameters 'win.size' and step.size are ignored and the statistics are estimated for 'grfeatures'.

`stat` Statistic used to estimate the summarized value of the variable of interest in each interval/window. Posible options are: "mean", geometric mean ("gmean"), "median", "density", and "sum" (default). Here, we define "density" as the sum of values from the variable of interest in the given region devided by the length of the region.

`absolute` Optional. Logic (default: FALSE). Whether to use the absolute values of the variable provided

`select.strand` Optional. If provided,"+" or "-", then the summarized statistic is computed only for the specified DNA chain.

`column` Integer number denoting the column where the variable of interest is located in the metacolumn of the GRanges object or an integer vector of two elements (only if prob = TRUE).

`prob` Logic. If TRUE and the variable of interest has values between zero and 1, then the summarized statistic is comuputed using Fisher's transformation. If length(column) == 2, say with colums x1 and x2, then the variable of interest will be p = x1/(x1 + x2). For example, if x1 and x2 are methylated and unmethylated read counts, respectively, then p is the methylation level.

`entropy` Logic. Whether to compute the entropy when prob == TRUE.

`maxgap, minoverlap, type`
See ?findOverlaps in the IRanges package for a description of these arguments.

`scaling` integer (default 1). Scaling factor to be used when stat = "density". For example, if scaling = 1000, then density * scaling denotes the sum of values in 1000 bp.

`logbase` A positive number: the base with respect to which logarithms

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

`na.rm` Logical value. If TRUE, the NA values will be removed

`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

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

## Value

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

## Author(s)

Robersy Sanchez

## Examples

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

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

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

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

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

---

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

---

## Description

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

## Usage

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

## Arguments

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

## Value

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

### Examples

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

## hclust_rect *Draw Rectangles with Background Colors Around Hierarchical Clusters*

### Description

Draws rectangles with background colors around the branches of a dendrogram highlighting the corresponding clusters. First the dendrogram is cut at a certain level, then a rectangle is drawn around selected branches.

### Usage

```
hclust_rect(tree, k = NULL, which = NULL, x = NULL, h = NULL,
  border = 2, cluster = NULL, cuts = NULL, color = NULL, ...)
```

### Arguments

`tree` The same as in `rect.hclust`

`k, h` The same as in `rect.hclust`

`which, x` The same as in `rect.hclust`

`border` The same as in `rect.hclust`

`cluster` The same as in `rect.hclust`

`cuts` A numeric vector used to manually locate the rectangles around hierarchical clusters in the rigth position. This is tricky since each experimental dataset yield different measurement scale and must be manually adjusted. Settings are cuts = c(xleft, ybottom, xright, ytop). Default is NULL. Use it only if need it.

`color` Background color to use inside the rectangles around hierarchical clusters. Default is NULL.

### Details

This function is exactly function `rect.hclust` with a nice feature added: "to draw the rectangles around hierarchical clusters with background colors".

## Examples

```
### Violent crime rates by US state
hca <- hclust(dist(USArrests))

# Basic key RGB colors
# rgb(red, green, blue, alpha, names = NULL, maxColorValue = 1)
clusters.color = c(rgb(0, 0.7, 0, 0.1), rgb(0, 0, 1, 0.1),
                   rgb(1, 0.2, 0, 0.1))

plot(hca)
hclust_rect(hca, h = 150,  border = clusters.color, col = clusters.color )
```

---

# jensenSDiv *Compute Jensen-Shannon Divergence*

---

## Description

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

## Usage

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

## Arguments

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

## Details

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

## References

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

### Examples

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

## ksTest *Kolmogorov-Smirnov statistics*

### Description

Permutation test for Kolmogorov-Smirnov statistics

### Usage

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

### Arguments

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

### Value

gamma distribution CDF

### Author(s)

Robersy Sanchez - 02/29/2016

### References

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

### Examples

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

## predict.GammaMixt *Predict function for the DMP's Mixtures of Gamma Distributions model*

### Description

This is an utility function to get the density, probailities, and posterior probability predictions based on a given DMP's Mixtures of Gamma Distributions (GMD) model, obtained with function gammaMixtCut.

### Usage

```
## S3 method for class 'GammaMixt'
predict(gmd, pred = "quant", q = 0.95,
  div.col = NULL, interval = NULL)
```

### Arguments

`gmd` An object carrying the best nonlinear fit for a distribution model obtained with function nonlinearFitDist ('GammaMixt' class).

`pred` Type of prediction resquested: *density* ("dens"),*quantiles* ("quant"), *random number* ("rnum"), *probabilities* ("prob"), or classification *posterior probability* ("postPrb").

`q` numeric vector of quantiles, probabilities or an interger if pred = "rnum", or A "pDMP"or "InfDiv" object obtained with functions getPotentialDIMP or estimateDivergence. These are list of GRanges objects, where each GRanges object from the list must have at least two columns: a column containing the total variation of methylation level (TV, difference of methylation levels) and a column containing a divergence of methylation levels (it could be TV or Hellinger divergence).

`div.col` An integer. If 'q' is "pDMP"or "InfDiv" object, then it is the column number for the divergence of methylation levels used in the estimation of model 'gmd'. Default: NULL.

`interval` a vector containing the end-points of the interval to be searched for the quantile(s). An interval would be, e.g., 'interval = c(min(x), max(x))', where 'x' is the variable used to estimate model 'gmd'.

`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*.

---

`predict.ProbDistr` *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 'ProbDistr'
predict(nlm, pred = "quant", q = 0.95, dist.name)

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

## Arguments

`nlm` 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").

`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

```
set.seed(1)
num.points <- 1000
HD <- makeGRangesFromDataFrame(
  data.frame(chr = "chr1", start = 1:num.points, end = 1:num.points,
             strand = '*',
             hdiv = rweibull(1:num.points, shape = 0.75, scale = 1)),
  keep.extra.columns = TRUE)
nlms <- nonlinearFitDist(list(HD), column = 1, verbose = FALSE)

x=seq(0.1, 10, 0.05)
y <- predict(nlms[[1]], pred="dens", q = x,
                dist.name="Weibull2P")
y1 <- dweibull(x, shape = 0.75, scale = 1)
# The maximum difference between the "theoretical" and estimated densities
max(abs(round(y, 2) - round(y1, 2)))
```

---

# `predictDIMPclass` *Predict DIMP class*

---

## Description

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

## Usage

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

## Arguments

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

## Details

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

## Value

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

## Examples

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

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

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

### Description

Beta Regression analysis for treatment versus control group comparison of methylation levels, appends three new metacolumns "beta", "log2FC", "pvalue" to the provided GRanges argument

### Usage

```
propTest(GR, control.names, treatment.names, link = "logit",
  type = "ML", tv.cut = NULL, indvPerGrp = 0, FilterLog2FC = TRUE,
  pAdjustMethod = "BH", pvalCutOff = 0.05, Minlog2FC = 0.5,
  saveAll = FALSE, num.cores = 1, tasks = 0L, verbose = TRUE)
```

### Arguments

`GR` GRanges objects including control and treatment samples containing the methylation levels. The name for each column must coincide with the names given for parameters: ’control.names’ and ’treatment.names’.

`control.names` Names/IDs of the control samples, which must be include in the variable GR at the metacolumn.

`treatment.names` Names/IDs of the treatment samples, which must be included in the variable GR at the metacolumn.

`link` Parameter to be passed to function ’betareg’ from package ’betareg’. character specification of the link function in the mean model (mu). Currently, "logit", "probit", "cloglog", "cauchit", "log", "loglog" are supported. Alternatively, an object of class "link-glm" can be supplied (see betareg).

`type` Parameter to be passed to function ’betareg’ from package ’betareg’. A character specification of the type of estimator. Currently, maximum likelihood ("ML"), ML with bias correction ("BC"), and ML with bias reduction ("BR") are supported.

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

`indvPerGrp` An integer number giving the minimum number of individuals per group at each site/region. Default 2.

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

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

`pvalCutOff` cutoff used, then a p-value adjustment is performed. If NULL all the reported p-values are for testing.

`Minlog2FC` minimum logarithm base 2 of fold changes.

`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

## Details

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

## Value

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

## Examples

```
num.cyt <- 11001 # Number of cytosine position with methylation call
max.cyt = 14000
## Gene annotation
genes <- GRanges(seqnames = "1",
                 ranges = IRanges(start = c(3631, 6788, 11649),
                                  end = c(5899, 9130, 13714)),
                 strand = c("+", "-", "-"))
mcols(genes) <- data.frame(gene_id = c("AT1G01010", "AT1G01020",
                                       "AT1G01030"))

set.seed(123) #'#' To set a seed for random number generation
## GRanges object of the reference with methylation levels in
## its meta-column
Ref <- makeGRangesFromDataFrame(
  data.frame(chr = '1',
             start = 3000:max.cyt,
             end = 3000:max.cyt,
             strand = '*',
             p1 = rbeta(num.cyt, shape1 = 1, shape2 = 1.5)),
  keep.extra.columns = TRUE)

## List of Granges objects of individuals methylation levels
Indiv <- GRangesList(
  sample11 = makeGRangesFromDataFrame(
    data.frame(chr = '1',
               start = 3000:max.cyt,
               end = 3000:max.cyt,
               strand = '*',
               p2 = rbeta(num.cyt, shape1 = 1.5, shape2 = 2)),
    keep.extra.columns = TRUE),
  sample12 = makeGRangesFromDataFrame(
    data.frame(chr = '1',
               start = 3000:max.cyt,
               end = 3000:max.cyt,
               strand = '*',
               p2 = rbeta(num.cyt, shape1 = 1.6, shape2 = 2.1)),
    keep.extra.columns = TRUE),
  sample21 = makeGRangesFromDataFrame(
    data.frame(chr = '1',
               start = 3000:max.cyt,
               end = 3000:max.cyt,
               strand = '*',
               p2 = rbeta(num.cyt, shape1 = 10, shape2 = 4)),
    keep.extra.columns = TRUE),
  sample22 = makeGRangesFromDataFrame(
    data.frame(chr = '1',
               start = 3000:max.cyt,
               end = 3000:max.cyt,
               strand = '*',
               p2 = rbeta(num.cyt, shape1 = 11, shape2 = 4)),
    keep.extra.columns = TRUE))
## To estimate Hellinger divergence using only the methylation levels.
HD <- estimateDivergence(ref = Ref, indiv = Indiv, meth.level = TRUE,
                         columns = 1)
## To perform the nonlinear regression analysis
nlms <- nonlinearFitDist(HD, column = 4, verbose = FALSE)

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

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

## Finally DIMPs statistics genes
p_DIMPs = getGRegionsStat(GR = DIMPs, grfeatures = genes, stat = "mean",
                          prob = TRUE, column = 2L)
```

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

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

---

rmstGR *Root Mean Square Test for metadata in a list of GRanges objects*

---

### Description

Count data in MethylIT pipeline is carried in GRanges objects. This function provides a shortcut to apply the parametric Bootstrap of 2x2 Contingency independence test, which is implemented in function bootstrap2x2. The goodness of fit statistic is the root-mean-square statistic (RMST) or Hellinger divergence, as proposed by Perkins et al. [1, 2]. Hellinger divergence (HD) is computed as proposed in [3].

### Usage

```
rmstGR(LR, count.col = 1:2, control.names = NULL,
  treatment.names = NULL, stat = "rmst", pooling.stat = "sum",
  tv.cut = NULL, hdiv.cut = NULL, hdiv.col = NULL,
  num.permut = 100, pAdjustMethod = "BH", pvalCutOff = 0.05,
  saveAll = FALSE, num.cores = 1L, tasks = 0L, verbose = TRUE, ...)
```

### Arguments

LR
: A list of GRanges, a GRangesList, a CompressedGRangesList object. Each GRanges object from the list must have two columns: methylated (mC) and unmethylated (uC) counts. The name of each element from the list must coincide with a control or a treatment name.

count.col
: 2d-vector of integers with the indexes of the read count columns. If not given, then it is asssumed that the methylated and unmethylated read counts are located in columns 1 and 2 of each GRanges metacolumns. If object LR is the output of Methyl-IT function estimateDivergence, then columns 1:4 are the read count columns: columns 1 and 2 are methylated and unmethylated read counts from the reference group, while columns 3 and 4 are methylated and unmethylated read counts from the treatment group, respectively. In this case, if the requested comparison is reference versus treatment, then no specification is needed for count.col. The comparison control versus treatment can be obtained by setting count.col = 3:4 and providing control.names and treatment.names.

control.names, treatment.names
: Names/IDs of the control samples, which must be included in the variable GR at the metacolumn. 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.

`stat` Statistic to be used in the testing: 'rmst' (root mean square test) or 'hdiv' (Hellinger divergence test).

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

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

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

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

`num.permut` Number of permutations.

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

`pvalCutOff` cutoff used when a p-value adjustment is performed

`saveAll` if TRUE all the temporal results are returned

`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).

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

`...` Additional parameters for function `MethylIT`.

## Details

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

## Value

A GRanges object with the original sample counts, bootstrap p-value probability, total variation (difference of methylation levels), and p-value adjusment.

## References

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

## See Also

FisherTest

## Examples

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

rmstGR(LR = data, control.names = c("C1", "C2"),
       treatment.names = c("T1", "T2"),
       tv.cut = 0.25, num.permut = 100, pAdjustMethod="BH",
       pvalCutOff = 0.05, num.cores = 4L, verbose=TRUE)
```

## shannonEntr *Compute Shannon Entropy*

### Description

Compute Shannon Entropy of probability vector p.

### Usage

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

### Arguments

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

### Details

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

### Examples

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

## simulateCounts *Simulate read counts of methylated and unmethylated cytosines*

### Description

Auxiliary function to simulate read counts of methylated and unmethylated cytosines

### Usage

```
simulateCounts(num.samples, sites, alpha, beta, size, theta,
  sample.ids = NULL)
```

## Arguments

| | |
|---|---|
| `num.samples` | Number of samples to generate. |
| `sites` | Number of cytosine sites for each sample. |
| `alpha` | Alpha parameter of beta distribution. Parameter shape1 from `Beta` function. |
| `beta` | Beta parameter of beta distribution. Parameter shape2 from `Beta` function. |
| `size` | number of trials (11 or more). Expected cytosine coverage. |
| `theta` | Parameter theta from `rnegbin` (overdispersion parameter). |
| `sample.ids` | Names for the samples. |

## Details

Methylation coverages (minimum 10) are generated from a Negative Binomial distribution with function `rnegbin` from R package MASS. This function uses the representation of the Negative Binomial distribution as a continuous mixture of Poisson distributions with Gamma distributed means. Prior methylation levels are randomly generated with beta distribution using `Beta` function from R package “stats” and posterior methylation levels are generated according Bayes’ theorem. The read of methylation counts are obtained as the product of coverage by the posterior methylation level.

## Value

A list of GRanges objects with the methylated and unmethylated counts in its metacolumn.

## Author(s)

Robersy Sanchez

## Examples

```
# *** Simulate samples with expected average of difference of methylation
# levels equal to 0.0427.
# === Expected mean of methylation levels ===
bmean <- function(alpha, beta) alpha/(alpha + beta)
bmean(0.03, 0.5) - bmean(0.007, 0.5) #' Expected difference = 0.04279707

# === The number of cytosine sitesto generate ===
sites = 5000
# == Set a seed for pseudo-random number generation ===
set.seed(123)

# === Simulate samples ===
ref = simulateCounts(num.samples = 1, sites = sites, alpha = 0.007,
                    beta = 0.5, size = 50, theta = 4.5, sample.ids = "C1")
treat = simulateCounts(num.samples = 2, sites = sites, alpha = 0.03,
                    beta = 0.5, size = 50, theta = 4.5,
                    sample.ids = c("T1", "T2"))

#  === Estime Divergences ===
HD = estimateDivergence(ref = ref$C1, indiv =  treat, Bayesian = TRUE,
                        num.cores = 1L, percentile = 1)

# === Difference of methylation levels of treatment simulated samples.
# Treatment versus reference
data.frame(mean.diff = c(mean(HD$T1$TV), mean(HD$T2$TV)),
           c("T1", "T2"), row.names = 2)
```

# Index