Package 'epigenomix'

November 19, 2019

Type Package

Title Epigenetic and gene transcription data normalization and integration with mixture models

Version 1.26.0

Date 2017-04-09

Author Hans-Ulrich Klein, Martin Schaefer

Maintainer Hans-Ulrich Klein <h.klein@uni-muenster.de>

Depends R (>= 3.2.0), methods, Biobase, S4Vectors, IRanges, GenomicRanges, SummarizedExperiment

Imports BiocGenerics, MCMCpack, Rsamtools, parallel, GenomeInfoDb, beadarray

Description A package for the integrative analysis of RNA-seq or microarray based gene transcription and histone modification data obtained by ChIP-seq. The package provides methods for data preprocessing and matching as well as methods for fitting bayesian mixture models in order to detect genes with differences in both data types.

License LGPL-3

biocViews ChIPSeq, GeneExpression, DifferentialExpression, Classification

git_url https://git.bioconductor.org/packages/epigenomix

git_branch RELEASE_3_10

git_last_commit ef69184

git_last_commit_date 2019-10-29

Date/Publication 2019-11-18

R topics documented:

`bayesMixModel` *Fits a Bayesian mixture model using Markov Chain Monte Carlo (MCMC) methods*

Description

This method estimates the posterior distribution of a Bayesian mixture model using Markov Chain Monte Carlo (MCMC) methods and calculates measures of this distribution. The mixture model may consist of normal components (with a fixed expectation of 0), exponential components and gamma components, which may be mirrored in order to model negative values.

Usage

```
bayesMixModel(z, normNull=c(), expNeg=c(), expPos=c(), gamNeg=c(), gamPos=c(), sdNormNullInit=c(),
```

Arguments

`z`	Observed values
`normNull`	Indices of the normal components (that have $\mu = 0$).
`expNeg`	Indices of the mirrored exponential components.
`expPos`	Indices of the exponential components.
`gamNeg`	Indices of the mirrored gamma components.
`gamPos`	Indices of the gamma components.
`sdNormNullInit`	Initial standard deviations of the normal components.
`rateExpNegInit`	Initial rates of the mirrored exponential components. Only relevant if mirrored exponential components are specified.
`rateExpPosInit`	Initial rates of the exponential components. Only relevant if exponential components are specified.
`shapeGamNegInit`	Initial shape parameters of the mirrored gamma components. Only relevant if mirrored Gamma components are specified.

`scaleGamNegInit`
: Initial scale parameters of the mirrored gamma components. Only relevant if mirrored Gamma components are specified.

`shapeGamPosInit`
: Initial shape parameters of the gamma components. Only relevant if Gamma components are specified.

`scaleGamPosInit`
: Initial scale parameters of the gamma components. Only relevant if Gamma components are specified.

`piInit`
: Initial weights of the components. If missing, all k components get the same initial weight 1/k.

`classificationsInit`
: Initial classifications of the data points. If missing, all data points are assigned to class floor(k/2) with k = number of components.

`dirichletParInit`
: Initial concentration parameter of prior distribution assigned to the mixture weights.

`shapeDir`
: Prior shape parameter of Gamma distribution for concentration parameter of prior distribution assigned to the mixture weights.

`scaleDir`
: Prior scale parameter of Gamma distribution for concentration parameter of prior distribution assigned to the mixture weights.

`weightsPrior`
: Prior distribution assigned to mixture weights. Available are the Finite-dimensional Dirichlet prior ("FDD"), also known as Dirichlet-multinomial process, and the Truncated Dirichlet process ("TDP"). Both are approximations to the Dirichlet process for a large number of components, while the Finite-dimensional Dirichlet prior is also suited for a small number of components as a special case of the Dirichlet distribution.

`sdAlpha`
: Standard deviation of proposal distribution for concentration parameter of the prior distribution assigned to the mixture weights in the Metropolis-Hastings step incorporated in the Gibbs sampler. Only relevant if `weightsPrior="FDD"`.

`shapeNorm0`
: Prior shape parameter of Gamma distribution for precision of normal components.

`scaleNorm0`
: Prior scale parameter of Gamma distribution for precision of normal components.

`shapeExpNeg0`
: Prior shape parameter of Gamma distribution for parameter of mirrored exponential components. Only relevant if mirrored exponential components are specified.

`scaleExpNeg0`
: Prior scale parameter of Gamma distribution for parameter of mirrored exponential components. Only relevant if mirrored exponential components are specified.

`shapeExpPos0`
: Prior shape parameter of Gamma distribution for parameter of exponential components. Only relevant if exponential components are specified.

`scaleExpPos0`
: Prior scale parameter of Gamma distribution for parameter of exponential components. Only relevant if exponential components are specified.

`shapeGamNegAlpha0`
: Prior shape parameter of Gamma distribution for shape parameter of mirrored Gamma components. Only relevant if mirrored Gamma components are specified.

`shapeGamNegBeta0`
: Prior scale parameter of Gamma distribution for shape parameter of mirrored Gamma components. Only relevant if mirrored Gamma components are specified.

`scaleGamNegAlpha0`
: Prior shape parameter of Gamma distribution for scale parameter of mirrored Gamma components. Only relevant if mirrored Gamma components are specified.

`scaleGamNegBeta0`
: Prior scale parameter of Gamma distribution for scale parameter of mirrored Gamma components. Only relevant if mirrored Gamma components are specified.

`shapeGamPosAlpha0`
: Prior shape parameter of Gamma distribution for shape parameter of Gamma components. Only relevant if Gamma components are specified.

`shapeGamPosBeta0`
: Prior scale parameter of Gamma distribution for shape parameter of Gamma components. Only relevant if Gamma components are specified.

`scaleGamPosAlpha0`
: Prior shape parameter of Gamma distribution for scale parameter of Gamma components. Only relevant if Gamma components are specified.

`scaleGamPosBeta0`
: Prior scale parameter of Gamma distribution for scale parameter of Gamma components. Only relevant if Gamma components are specified.

`itb`
: Number of iterations used for burn-in.

`nmc`
: Number of iterations after burn-in used for analysis.

`thin`
: Thinning value for the iterations after burn-in.

`average`
: Way of averaging across the posterior distribution to obtain estimates of model parameters. Either `average="mean"` or `average="median"`. Note: For the allocation to components, results are given for posterior mean, median and maximum density regardless of the specification.

`sdShape`
: Standard deviation of proposal distribution for shape parameter of Gamma components in the Metropolis-Hastings step incorporated in the Gibbs sampler. Only relevant if Gamma components are specified.

Details

The convergence of Markov chains must be assessed prior to an interpretation of results. Inspection of trace plots via `plotChains` is therefore urgently recommended. Iterations during which one of the chains has not yet reached stationarity should not be taken into account for analysis and can be excluded by setting an appropriate burn-in value `itb`. Autocorrelation between subsequent chain values can be reduced by thinning the chain, setting an appropriate value for `thin`. To ensure a sufficient number of iterations for the chains after the burn-in, `nmc` should be increased when the thinning is increased. The standard deviations of the proposal distribution in a Metropolis-Hastings step should be tuned to achieve a medium-level acceptance rate (e.g., 0.3-0.7): A very low acceptance rate would cause a long running time of the algorithm, while a very high acceptance rate typically leads to autocorrelation between the values of the chain. Acceptance is documented for each iteration in the `chains` slot of objects of class `MixModelBayes-class`.

Value

An object of class MixModelBayes-class storing results, data, priors, initial values and information about convergence.

Author(s)

Martin Schaefer (martin.schaefer@udo.edu)

See Also

plotChains, MixModelBayes-class

Examples

```
set.seed(1000)
z <- c(rnorm(1000, 0, 0.5), rnorm(1000, 0, 1))
mm <- bayesMixModel(z, normNull=1:2, sdNormNullInit=c(0.1, 0.2),
  piInit=c(1/2, 1/2), shapeNorm0=c(1, 1), scaleNorm0=c(1, 1),
  shapeExpNeg0=c(), scaleExpNeg0=c(),
  shapeExpPos0=c(), scaleExpPos0=c(), sdAlpha=1, itb=100, nmc=1000, thin=10)
mm
plotComponents(mm)
plotChains(mm, chain="pi")

z <- c(rnorm(200, 0, 1), rnorm(200, 0, 5), rexp(200, 0.1), -rexp(200, 0.1))
mm <- bayesMixModel(z, normNull=1:2, gamNeg=3, gamPos=4,
  sdNormNullInit=c(1, 1),
  shapeGamNegInit=1, scaleGamNegInit=1, shapeGamPosInit=1, scaleGamPosInit=1,
  shapeNorm0=c(1,3), scaleNorm0=c(1,3), sdAlpha=1,
  shapeGamNegAlpha0=1, shapeGamNegBeta0=1,
  scaleGamNegAlpha0=1, scaleGamNegBeta0=1,
  shapeGamPosAlpha0=1, shapeGamPosBeta0=1,
  scaleGamPosAlpha0=1, scaleGamPosBeta0=1, sdShape=0.025,
  itb=100, nmc=1000, thin=10)
mm
plotComponents(mm)
plotChains(mm, chain="pi")
```

---

calculateCrossCorrelation

Calculate the cross correlation for a given GRanges object.

Description

This method calculates the cross correlation, i.e. the Pearson correlation between the coverages of the positive and negative strand from a DNA sequencing experiment. The cross correlation can be used as a quality measure in ChIP-seq experiments (Kharchenko et al. 2008). Cross correlation can also be used to estimate the fragment size by determining the shift (given in base pairs) that maximizes the cross correlation.

Usage

```
calculateCrossCorrelation(object, shift=c(200,250,300), bin=10, mode="none", minReads=10000, chrs=
```

Arguments

object	An GRanges object containing the aligned reads.
shift	The number of bases that the negative strand is shifted towards its three prime end. This can be a vector, if the correlation should be calculated for different shifts.
bin	If bin is larger than one, the coverage is calculated for bins of size bin and not for each single base. This speeds up calculations and might be beneficial in cases of low coverage. Note that shifting is performed after binning, so that the shift(s) should be a multiple of bin (otherwise, shift is rounded to the nearest multiple of bin).
mode	mode defines how bases (or bins) without reads are handled. both means that only bases covered on both strands are included when calculating the correlation. one means that the base has to be covered on at least one strand and none mean that all bases are included independent of their coverage.
minReads	If not at least minReads are mapped to a chromosome, the chromosome is omitted.
chrs	A character vector with the chromosomes that should be included into the calculation. NA means all chromosomes.
mc.cores	Number of cores to be used.

Details

Only 5 prime start positions of reads are used for calculating the coverage. Therefore, after removing duplicates in a single end sequencing experiment, the coverage can not be larger than one, if the bin size is set to one. (In this setting, mode both is meaningless.) If bin is larger than one, the coverage within a bin is aggregated. Then, the correlation is calculated for each shift. A shift (given in basepairs) should be multiple of the bin size (given in basepairs, too). If not, the binnend coverage is shifted by round(shift/bin) elements.

The different modes define whether regions without coverage or with only one covered strand should used. The original implementation in the package "spp" does not make use of regions without coverage. However, this seems to be a loss of information, since no coverage has also a biological meaning in a ChIP-seq experiment. If the fragment size is approximately 500bp, setting shift=seq(200,800,10), bin=10 and mode="none" should be a good setting.

After the cross correlation was calculated for each chromosome, the weighted mean correlation across all chromosomes is calculated. The weight for a specific chromosome equals the fraction of all reads that were aligned to that chromosome.

Value

A numeric vector with the cross correlation for each shift. The names of the vector correspond to the shifts.

Author(s)

Hans-Ulrich Klein (hklein@broadinstitute.org)

References

Kharchenko PV, Tolstorukov MY and Park PJ. Design and analysis of ChIP-seq experiments for DNA-binding proteins. Nat Biotechnol 2008, 26(12):1351-9

Landt SG et al., ChIP-seq guidelines and practices of the ENCODE and modENCODE consortia. Genome Res. 2012, 22(9):1813-31

See Also

GRanges-class

Examples

```
triangularKernel <- function(x, pos, h)  {
  res <- 1 - (abs(x - pos) / h)
  res[res < 0] <- 0
  return(res)
}
covPos <- round(triangularKernel(1:100, 60, 50) * 100)
covNeg <- round(triangularKernel(1:100, 65, 50) * 100)

reads <- GRanges(IRanges(start=c(rep(seq_along(covPos), covPos), rep(seq_along(covNeg), covNeg) - 9),
                         width=10),
                         strand=c(rep("+", sum(covPos)), rep("-", sum(covNeg))),
                         seqnames=rep("1", sum(covPos)+sum(covNeg)))

calculateCrossCorrelation(reads, shift=c(0,10), bin=1, mode="none", minReads=1)
cor(covPos, covNeg)
cor(covPos[1:(length(covPos)-10)], covNeg[11:length(covNeg)])

calculateCrossCorrelation(reads, shift=c(0,10), bin=1, mode="one", minReads=1)
cor(covPos[covPos != 0 | covNeg != 0], covNeg[covPos != 0 | covNeg != 0])

calculateCrossCorrelation(reads, shift=c(0,10), bin=1, mode="both", minReads=1)
cor(covPos[covPos != 0 & covNeg != 0], covNeg[covPos != 0 & covNeg != 0])


covPos2 <- round(triangularKernel(1:100, 60, 50) * 50)
covNeg2 <- round(triangularKernel(1:100, 68, 50) * 50)
reads2 <- GRanges(IRanges(start=c(rep(seq_along(covPos2), covPos2), rep(seq_along(covNeg2), covNeg2) - 9),
                          width=10),
                          strand=c(rep("+", sum(covPos2)), rep("-", sum(covNeg2))),
                          seqnames=rep("2", sum(covPos2)+sum(covNeg2)))
seqlevels(reads2) <- c("1", "2")
allReads <- c(reads, reads2)

calculateCrossCorrelation(allReads, shift=5, minReads=1, bin=1, mode="none")
cor1 <- cor(covPos[1:(length(covPos)-5)], covNeg[6:length(covNeg)])
cor2 <- cor(covPos2[1:(length(covPos2)-5)], covNeg2[6:length(covNeg2)])
cor1 * (sum(c(covPos, covNeg))/length(allReads)) +
  cor2 * (sum(c(covPos2, covNeg2))/length(allReads))
```

ChIPseqSet-class *Class* "ChIPseqSet"

Description

A class for storing count data obtained from ChIP-seq experiments by counting the number of reads lying within regions. The class extends RangedSummarizedExperiment.

Objects from the Class

Objects can be created by calls of the form `ChIPseqSet(chipVals=countDataMatrix,rowRanges=genomicRegions,c`
However, one will most likely create a `ChIPseqSet` object by calling `summarizeReads`.

Slots

`metadata`: An optional `list` of arbitrary content describing the overall experiment.

`rowRanges`: Object of class `"GRanges"` or `"GRangesList"` containing the genomic regions where the reads were counted.

`colData`: Object of class `"DataFrame"` containing information on variable values of the samples. Some methods require the total library size of each sample stored in a column titled totalCounts.

`assays`: Object of class `SimpleList` of a matrix, named `chipVals` containing the read counts per genomic region.

Extends

Class `"RangedSummarizedExperiment"`, directly.

Methods

chipVals `signature(object = "ChIPseqSet")`: Returns the matrix with read counts.

chipVals<- `signature(object = "ChIPseqSet",value = "matrix")`: Sets the matrix with read counts.

cpm `signature(object = "ChIPseqSet",libSize,log2=FALSE,priorCount=0.1)`: Returns an object of `ChIPseqSet` with read counts standardized by library size - counts per million (cpm). If the library size is not given, the column sums of the given object are used. Cpm values are logarithmized after adding `priorCounts`, if log2 is `TRUE`.

Author(s)

Hans-Ulrich Klein (hklein@broadinstitute.org)

See Also

`summarizeReads`, `normalizeChIP`

Examples

```
showClass("ChIPseqSet")
```

eSet *Example gene expression data set.*

Description

The ExpressionSet stores 2 replicates for each of two different conditions. Data were obtained from Affymetrix MouseGene 1.0 ST arrays.

Usage

```
data(eSet)
```

Format

An object of class ExpressionSet.

Details

The example data contains a subset of 200 probesets located on chromosome 1. Data were RMA normalized.

Examples

```
data(eSet)
eSet
pData(eSet)
```

fpkm *Example RNA-seq data set.*

Description

The data.frame stores transcription values obtained from the Cufflinks software for two samples (CEBPA_WT and CEBPA_KO). Transcription values are given in fragments per kilobase of transcripts per million fragments (FPKM).

Usage

```
data(fpkm)
```

Format

An object of class data.frame.

Details

All transcripts sharing the TSS were grouped and one transcription values is given for each group of transcripts. The example data contains a subset of about 3500 TSS located on chromosome 1.

Examples

```
data(fpkm)
head(fpkm)
```

`getAlignmentQuality` *Calculation of basic alignments statistics*

Description

Calculates some basic alignment statistics for given bam files.

Usage

```
getAlignmentQuality(bamFile, verbose = FALSE, mc.cores = 1)
```

Arguments

`bamFile`	A `character` vector with the filenames of the bam files
`verbose`	If set to `TRUE`, some status information is written to the R console.
`mc.cores`	Number of cores to be used.

Details

The given bam files should have marked duplicates and not uniquely mapped reads should have a quality value of 0. In detail, this function returns a data frame with the following columns:

Sample File name without path and suffix

HeaderID ID field from bam header, if available

HeaderSampleID SM field from bam header, if available

HeaderLibraryID LB field from bam header, if available

TotalReads Total number of reads in bam file

MappedReads Number of mapped Reads

MappedReadsRel MappedReads/TotalReads

UniquelyMappedReads Number of mapped reads with mapping quality larger 0

UniquelyMappedReadsRel UniquelyMappedReads/MappedReads

UniquelyMappedUniqueReads Number of non duplicated mapped reads with mapping quality larger 0

UniquelyMappedUniqueReadsRel UniquelyMappedUniqueReads/MappedReads

NonRedundantFraction UniquelyMappedUniqueReads/UniquelyMappedReads

QualMean Mean mapping quality of all uniquely mapped unique reads

QualSd Standard deviation of the mapping quality of all uniquely mapped unique reads

Quantile0 0% quantileof the mapping quality of all uniquely mapped unique reads

Quantile25 25% quantile of the mapping quality of all uniquely mapped unique reads

Quantile50 50% quantile of the mapping quality of all uniquely mapped unique reads

Quantile75 75% quantile of the mapping quality of all uniquely mapped unique reads

Quantile100 100% quantile of the mapping quality of all uniquely mapped unique reads

Path Full path and file name as given in argument `bamFile`

Value

Returns a data frame with one row for each given bam file and the columns as listed in the details section.

Author(s)

Hans-Ulrich Klein (hklein@broadinstitute.org)

Examples

```
## Not run: getAlignmentQuality("myFile.bam")
```

`integrateData` *Calculates a normalized correlation score from ChIP-seq and microarray gene expression data.*

Description

This function calculates the product of the standardized differences between two conditions in ChIP-seq data and the respective standardized differences in gene expression data. A score close to zero means that there are no (large) differences in at least one of the two data sets. If the score is positive, equally directed differences exist in both data sets. In case of a negative score, differences have unequal signs in the two data sets.

Usage

```
integrateData(expr, chipseq, factor, reference)
```

Arguments

`expr`	An `ExpressionSet` holding the gene expression data.
`chipseq`	A `ChIPseqSet` holding the ChIP-seq data.
`factor`	A `character` giving the name of the factor that describes the conditions to be compared. The factor must be present in the pheno data slot of the objects expr and chipseq. Further, the factor must have exactly two levels and the level names must be the same in both objects.
`reference`	Optionally, the name of the factor level that should be used as reference. If missing, the first level of `factor` in the object `expr` is used.

Details

Let A and B denote the gene expression value of one probe in the group of interest and in the reference group defined by the argument `reference`. And let X and Y be the ChIP-seq values assigned to that probe. This functions returnes for each probe

$$Z = (A - B)/\sigma_{ge} \times (X - Y)/\sigma_{chip},$$

where σ_{ge} is the standard deviation estimated from all observed difference in the gene expression data and σ_{chip} the standard deviation in the ChIP-seq data.

If there is more than one sample in any group and data set, the average of the replicates is calcuated first and than plugged into the formula above.

Not all features in `expr` must also be in `chipseq` and vice versa. Features present in only one of the two data types are omitted.

Value

A matrix with five columns. The first 4 columns store the (average) expression values and the (average) ChIP-seq values for each of the two conditions. The fith columns store the correlation score. The row names equal common feature names of `expr` and `chipseq`.

Author(s)

Hans-Ulrich Klein (h.klein@uni-muenster.de)

See Also

`summarizeReads` `normalizeChIP`

Examples

```
ge <- matrix(c(5,12,5,11,11,10,12,11), nrow=2)
row.names(ge) <- c("100_at", "200_at")
colnames(ge) <- c("c1", "c2", "t1", "t2")
geDf <- data.frame(status=c("control", "control", "treated", "treated"),
  row.names=colnames(ge))
eSet <- ExpressionSet(ge, phenoData=AnnotatedDataFrame(geDf))

chip <- matrix(c(10,20,20,22), nrow=2)
row.names(chip) <- c("100_at", "200_at")
colnames(chip) <- c("c", "t")
rowRanges <- GRanges(IRanges(start=c(10,50), end=c(20,60)), seqnames=c("1","1"))
names(rowRanges) = c("100_at", "200_at")
chipDf <- DataFrame(status=factor(c("control", "treated")),
  totalCount=c(100, 100),
  row.names=colnames(chip))
cSet <- ChIPseqSet(chipVals=chip, rowRanges=rowRanges, colData=chipDf)

integrateData(eSet, cSet, factor="status", reference="control")
```

---

mappedReads *Mapped reads obtained from a anti-histone ChIP-seq experiment.*

Description

The `GRangesList` contains two elements: "CEBPA_WT_1" and "CEBPA_KO_1". Both list elements are `GRanges` objects storing mapped reads from anti-H3K4me3 ChIP-seq experiments. The first sample was a wild-type mouse cell line. The second sample was obtained from the same cell line after CEPBA knock-out.

Usage

```
data(mappedReads)
```

Format

A `GRangesList` with two `GRanges`.

Details

Duplicated reads and reads mapping to more than one genomic location were removed. Reads were extended to the estimated DNA fragment size of 200bp towards the 3 prime end. Further, only reads lying within certain regions of chromomse 1 were kept to reduce storage space.

Examples

```
data(mappedReads)
names(mappedReads)
mappedReads[[1]]
```

---

matchProbeToPromoter *A function assigning promoter regions to given probe IDs.*

Description

This function returns a `GRangesList` object asigning promoter regions to probes. The assignment of transcripts to probes and the transcriptional start sites must be given as arguments.

Usage

```
matchProbeToPromoter(probeToTranscript, transcriptToTSS, promWidth = 4000, mode = "union", fix = "c
```

Arguments

`probeToTranscript`
: A `list` with character vectors as elements. The elements' names are probe IDs and the character vectors store the transcript IDs assigned to that probe.

`transcriptToTSS`
: A `data.frame` with four columns:

1. Transcript ID as given in the argument `probeToTranscript`
2. Chromosome
3. Transcriptional start site in base pairs
4. Strand

`promWidth`
: Width of the promoter regions in base pairs. Promoters are defined as `promWidth` base pairs upstream of the transcriptional start site. (default 4000bp)

`mode`
: How probes with multiple transcripts should be handled. Must be either "union", "keepAll" or "dropMultiple". (default "union")

`fix`
: Denotes what to use as anchor when defining the promoter region. Must be either "center", "start" or "end". "Center" means that the TSS is in the middle of the promoter, whereas "end" means that the promoter is placed upsream of the TSS. (default "center")

Details

More than one transcript can be assigned to one probe in the given `probeToTranscript` argument. Several options how to handle such cases can be choosen by argument `mode`. "union": The union of all promoters is calculated and assigned to the probe. "keepAll": All promoters of all transcripts are assigned to the probe. If some transcript have identical TSSs, the same promoter region occurs several times. "dropMultiple": All probes that have more than one transcript with different TSS are removed.

The argument `transcriptToTSS` must have at least 4 columns giving the information as described above. The column names are not decisive, but their position.

Value

An object of class `GRangesList` with one element for each probe. If `mode` is not set to "dropMultiple", `GRanges` may consist of more than one range. The names of the lists' elements are the probe IDs and additionally, each `GRanges` has a meta data column "probe" giving the corresponding probe ID.

Author(s)

Hans-Ulrich Klein (h.klein@uni-muenster.de)

See Also

`summarizeReads`

Examples

```
probeToTrans <- list("101"="ENST00011",
                     "102"=c("ENST00021", "ENST00022"),
                     "103"=NA)
transToTSS <- data.frame(
    transID=c("ENST00011", "ENST00021", "ENST00022"),
    chr=c("1", "1", "1"),
    tss=c(100000, 200000, 201000),
    strand=c("-", "+", "+"))

matchProbeToPromoter(probeToTrans, transToTSS,
    promWidth=4000, mode="union")
matchProbeToPromoter(probeToTrans, transToTSS,
    promWidth=4000, mode="keepAll")
```

MixModel-class *Class* "MixModel"

Description

This class stores a fitted mixture model.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

`mmData`: Object of class `"numeric"` storing the data.

`configuration`: Object of class `"list"` storing configuration. See notes for details.

`results`: Object of class `"list"` storing results. See notes for details.

Methods

as.data.frame `signature(object = "MixModel")`: Returns a data.frame containing the z-scores and classification results. The optional argument `classificationMethod` can be used to change the default classification method.

classification `signature(object = "MixModel",method = "character")`: Assess classification results.

classification `signature(object = "MixModel",method = "missing")`: Assess classification results.

components `signature(object = "MixModel")`: Assess mixture components.

mmData `signature(object = "MixModel")`: Assess data.

dim `signature(x = "MixModel")`: Assess dimension, i.e. numer of data points and number of components.

length `signature(x = "MixModel")`: Number of data points.

listClassificationMethods `signature(object = "MixModel")`: List available classification methods.

show `signature(object = "MixModel")`: Print an object of `MixModel` on screen.

summary `signature(object = "MixModel")`: Returns a list of data frames summarizing the parameter estimations for each component.

weights `signature(object = "MixModel")`: Asses the components weights.

Note

Slots `configuration` and `results` are lists with named elements. The following elements make up the minimum set of element that must be present. Depending on the method that was used to fit the mixture model, more elements may be present.

Slot `configuration` has at least one element.

1. inits A list with at least two elements: `component` and `pi`. `components` contains a list of objects of `MixtureComponent-class` storing the inital parameters of the mixture components. `pi` is a vector storing the initial components' weights.

Slot `results` has at least three elements.

1. components A list of objects of `MixtureComponent-class` storing the fitted mixture components.
2. pi A numeric vector holding the estimated components' weights.
3. classification A list of numeric vectors of the same length as `data` storing the classification results.

Author(s)

Hans-Ulrich Klein (h.klein@uni-muenster.de)

See Also

mlMixModel bayesMixModel MixModelML MixModelBayes

Examples

```
showClass("MixModel")
```

MixModelBayes-class *Class* "MixModelBayes"

Description

This class stores a Bayesian mixture model fitted by MCMC methods.

Objects from the Class

Objects can be created by calls of the form `new("MixModelBayes",...)`.

Slots

chains: Object of class "list" storing the course of the Markov chains for each parameter.

mmData: Object of class "numeric" storing the data.

configuration: Object of class "list" storing configuration. See notes for details.

results: Object of class "list" storing results. See notes for details.

Extends

Class "MixModel", directly.

Methods

chains `signature(object = "MixModelBayes")`: Gives access to the chains slot of the object.

acceptanceRate `signature(object = "MixModelBayes")`: Gives the acceptance rate for the parameter of the Dirichlet distribution. Acceptance rates between 0.3 and 0.7 are usually desired. Values not smaller than 0.1 (not larger than 0.9) might still be acceptable. The acceptance rate is only meaningful if the option weightsPrior was set to the Finite-dimensional Dirichlet prior.

Note

In addition to the content described in MixModel, the following elements are present: Slot configuration:

1. initsAs in MixModel.
2. priorsA list specifying the prior distributions for the parameters of the components and the parameter of the Dirichlet process.
3. chainA list with the technical specifications for the Markov Chains.

Slot results is exactly like in MixModel. Slot chains:

1. componentsA list giving the values for the parameters of the components in each iteration after burn-in and application of thinning.

2. piA matrix giving the values for the weights pi of the components in each iteration after burn-in and application of thinning.
3. dirichletParameterA vector giving the values for dirichlet Parameter in each iteration after burn-in and application of thinning.
4. classificationA matrix giving the number of genes classified to each components in each iteration after burn-in and application of thinning.

Author(s)

Hans-Ulrich Klein (h.klein@uni-muenster.de)

See Also

`bayesMixModel` `MixModel`

Examples

```
showClass("MixModelBayes")
```

MixModelML-class *Class* "MixModelML"

Description

This class stores a mixture model fitted by a maximum likelihood approach.

Objects from the Class

Objects can be created by calls of the form `new("MixModelML", ...)`. Usually, objects are created by `mlMixModel`.

Slots

`convergence`**:** Object of class `"list"` storing information about the convergence of the EM algorithm.

`mmData`**:** Object of class `"numeric"` storing the data.

`configuration`**:** Object of class `"list"` storing configuration. See notes for details.

`results`**:** Object of class `"list"` storing results. See notes for details.

Extends

Class `"MixModel"`, directly.

Methods

convergence `signature(object = "MixModelML")`: Access to the convergence information.

Note

In addition to the content described in MixModel, the following elements are present: Slot configuration:

1. convergenceA list storing the maximum number of allowed iterations. And delta log likelihood limit, that is interpreted as convergence, if the delta log likelihood falls below that limit.

Slot results is exactly like in MixModel. Slot convergence:

1. iterationsNumber of iterations ran.
2. deltaLogLikDelta of log likelihood observed in the last iteration.
3. logLikLog likelihood of the model fit.

Author(s)

Hans-Ulrich Klein (h.klein@uni-muenster.de)

See Also

mlMixModel MixModel

Examples

```
showClass("MixModelML")
```

MixtureComponent-class

Class "MixtureComponent"

Description

A class representing a mixture component.

Objects from the Class

Objects can be created by calls of the form new("MixtureComponent",...).

Slots

name: Object of class "character" giving the name or type of the mixture component.

parameters: Object of class "list" storing the parameters of corresponding distribution.

pdf: Object of class "function" giving the pdf of the mixture component.

color: Object of class "character" giving the color of the component that is used by plotting methods.

Methods

show signature(object = "MixtureComponent"): A method plotting a summary of the component on screen.

Note

The element in `parameters` should be named by the argument names of `pdf` such that this call works: `do.call(object@pdf,c(list(x=data),object@parameters))`

Author(s)

Hans-Ulrich Klein (h.klein@uni-muenster.de)

See Also

`MixModel`

Examples

```
showClass("MixtureComponent")
```

---

mlMixModel *Fits a mixture model using the maximum likelihood principle.*

Description

This method calculates the maximum likelihood estimations of a mixture model using the expectation-maximization (EM) algorithm. The mixture model may consists of normal components (with a fixed expectation of 0) and exponential components, which may be mirrored in order to model negative values.

Usage

```
mlMixModel(z, normNull = c(), expNeg = c(), expPos = c(), sdNormNullInit = c(), rateExpNegInit = c(),
```

Arguments

`z`	Observed values.
`normNull`	Indices of the normal components (that have $\mu = 0$).
`expNeg`	Indices of the mirrored exponential components.
`expPos`	Indices of the exponential components.
`sdNormNullInit`	Initial standard deviations of the normal components.
`rateExpNegInit`	Initial rates ("lambda") of the exponential components.
`rateExpPosInit`	Initial rates ("lambda") of the exponential components.
`piInit`	Initial weights of the components.
`maxIter`	Maximum number of iterations.
`tol`	Threshold for convergence. The minimum log likelihood gain between two iterations that must be achieved to continue.

Details

The EM algorithm is known to converge slowly in some cases and local maxima may avoid finding the optimal solution. Users should try different initial values and different convergence criteria.

The components' indices do not influence the result, but may influence the order in which components are listed or ploted by downstream methods. Indices must be successive integers from 1 to n.

Value

An object of MixModelML-class storing results, data, initial values and information about the convergence.

Author(s)

Hans-Ulrich Klein (h.klein@uni-muenster.de)

See Also

MixModelML-class

Examples

```
z <- c(rnorm(1000, 0, 0.5), rnorm(1000, 0, 1))
mm <- mlMixModel(z, normNull=1:2, sdNormNullInit=c(0.1, 0.2),
   pi=c(1/2, 1/2), maxIter=500, tol=0.001)
mm

z <- c(rnorm(1000, 0, 3), rnorm(1000, 0, 5), rexp(1000, 5), -rexp(1000, 5))
mm <- mlMixModel(z, normNull=1:2, expNeg=3, expPos=4,
   sdNormNullInit=c(1, 2), rateExpNegInit=8, rateExpPosInit=8,
   pi=c(1/4, 1/4, 1/4, 1/4), maxIter=500, tol=0.001)
mm
```

---

normalize *Normalization of ChIP-seq and other count data*

Description

This function implements some methods for between-sample normalization of count data. Although these methods were developed for RNA-seq data, they are also useful for ChIP-seq data normalization after reads were counted within regions or bins. Some methods may also be applied to count data after within-sample normalization (e.g. TPM or RPKM values).

Usage

```
## S4 method for signature 'ChIPseqSet'
normalize(object, method, isLogScale = FALSE, trim = 0.3, totalCounts)
## S4 method for signature 'ExpressionSet'
normalize(object, method, isLogScale = FALSE, trim = 0.3, totalCounts)
```

Arguments

object	An object of class ChIPseqSet or ExpressionSet that contains the raw data.
method	Normalization method, either "scale", "scaleMedianRegion", "quantile" or "tmm".
isLogScale	Indicates whether the raw data in object is already logarithmized. Default value is FALSE. Logarithmized data will be returned on the log scale, non logarithmized data will remain on its original scale.
trim	Only used if method is "tmm". Indicates the fraction of data points that should be trimmed before calculating the mean. Default value is 0.3.

`totalCounts` Only used if `method` is "scale". A vector giving the total number of reads for each sample. The Vector's length must equal the number of samples in `object`. Default values are the sums over all features for each sample (i.e. colsums of `object`).

Details

The following normalization methods are implemented:

1. scaleSamples are scaled by a factor such that all samples have the same number N of reads after normalization, where N is the median number of reads observed accross all samples. If the argument `totalCounts` is missing, the total numbers of reads are calculated from the given data. Otherwise, the values in `totalCounts` are used.
2. scaleMedianRegionThe scaling factor s_j for the j-th sample is defined as

$$s_j = median_i \frac{k_{ij}}{\prod_{v=1}^{m} k_{iv}}.$$

   k_{ij} is the value of region i in sample j. See Anders and Huber (2010) for details.
3. quantileQuantile normalization is applied to the ChIP-seq values such that each sample has the same cdf after normalization.
4. tmmThe trimmed mean M-value (tmm) normalization was proposed by Robinson and Oshlack (2010). Here, the logarithm of the scaling factor for sample i is calculated as the trimmed mean of

$$\log(k_{i,j}/m_j).$$

   Variable m_j denotes the geometric mean of region j. Argument `trim` is set to 0.3 as default value, so that the smallest 15% and the largest 15% of the log ratios are trimmed before calculating the mean.

Value

An object of the same class as the input `object` with the normalized data.

Author(s)

Hans-Ulrich Klein (hklein@broadinstitute.org)

References

Anders and Huber. Differential expression analysis for sequence count data. Genome Biol. 2010;11(10):R106.\
Robinson and Oshlack. A scaling normalization method for differential expression analysis of RNA-seq data. Genome Biol. 2010;11(3):R25

Examples

```
set.seed(1234)
chip <- matrix(c(rpois(20, lambda=10), rpois(20, lambda=20)), nrow=20,
               dimnames=list(paste("feature", 1:20, sep=""), c("sample1", "sample2")))
rowRanges <- GRanges(IRanges(start=1:20, end=1:20),
                     seqnames=c(rep("1", 20)))
names(rowRanges) = rownames(chip)
cSet <- ChIPseqSet(chipVals=chip, rowRanges=rowRanges)

tmmSet <- normalize(cSet, method="tmm", trim=0.3)
```

```
mean(log(chipVals(tmmSet))[, 1], trim=0.3) -
    mean(log(chipVals(tmmSet))[, 2], trim=0.3) < 0.01

quantSet <- normalize(cSet, method="quantile")
all(quantile(chipVals(quantSet)[, 1]) == quantile(chipVals(quantSet)[, 2]))
```

`normalizeChIP` *Normalization of ChIP-seq count data. (deprecated)*

Description

This method is deprecated. Use normalize instead. This method implements some normalization approaches for ChIP-seq data after counting reads within regions or bins. Similar methods are often applied to RNA-seq data after counting reads within genes.

Usage

```
normalizeChIP(object, method)
```

Arguments

`object`	A `ChIPseqSet` object as generated by `summarizeReads`
`method`	Normalization method, either "scaleTotal", "scaleRegion", "scaleMedianRegion" or "quantile"

Details

The following normalization methods are implemented:

1. scaleTotalSamples are scaled by a factor such that all samples have the same number of reads (the median number of reads observed accross all samples before normalization). All reads are used for calculating the scaling factor.
2. scaleRegionSamples are scaled by a factor such that all samples have the same number of reads (the median number of reads observed accross all samples before normalization). In contrast to scaleTotal, only reads falling into the regions (genes, promoters) that were used to create the given `ChIPseqSet` object are used for calculating the scaling factor. Hence, the sum of all columns of the returned `ChIPseqSet` are equal after applying this method.
3. scaleMedianRegionThe scaling factor s_j for the j-th sample is defined as:

$$s_j = median_i \frac{k_{ij}}{\prod_{v=1}^{m} k_{iv}}$$

   k_{ij} is the value of region i in sample j. See Anders and Huber (2010) for details.
4. quantileQuantile normalization is applied to the ChIP-seq values such that each sample has the same cdf after normalization.

Value

An `ChIPseqSet-class` object with normalized ChIP-seq values.

Author(s)

Hans-Ulrich Klein (h.klein@uni-muenster.de

References

Anders and Huber; Differential expression analysis for sequence count data; Genome Biology 2010, 11:R106

See Also

summarizeReads

Examples

```
chip <- matrix(c(5,6,5,6,10,12,10,12), nrow=4,
               dimnames=list(c("f1", "f2", "f3", "f4"), c("s1", "s2")))
rowRanges <- GRanges(IRanges(start=c(10, 20, 30, 40), end=c(11, 21, 31, 41)),
                     seqnames=c("1", "1", "1", "1"))
names(rowRanges) = rownames(chip)
chipDf <- DataFrame(totalCount=c(100, 100),
                    row.names=colnames(chip))
cSet <- ChIPseqSet(chipVals=chip, rowRanges=rowRanges, colData=chipDf)

chipVals(cSet)
chipVals(normalize(cSet, method="scaleMedianRegion"))
chipVals(normalize(cSet, method="quantile"))
```

---

plotChains *Produces trace plots for a Bayesian mixture model*

Description

This function method draws trace plots for a Bayesian mixture model, e.g. visualizes the course of the Markov Chains. Inspection of the Markov Chains is important to determine convergence of the chains, which is necessary for sensible results.

Usage

```
plotChains(object, chain, component, itb = 1, thin = 1, cols, ...)
```

Arguments

object	An object of MixModelBayes-class
chain	A character of length one giving the name of the paramter, which chain should be plotted. Can be omitted, if component is given. Then, all parameters of the given components are plotted.
component	An integer specifying the components, which parameter chains should be plotted. Can be omitted, if chain is given. Then, all trace plots are generated for all components having the parameter specified via argument chain.
itb	Number of iterations used for burn-in. The burn-in is relative to the output of bayesMixModel, e.g., any burn-in specified here is added to the burn-in that was specified when calling bayesMixModel.
thin	Thinning value for the iterations after burn-in. The thinning is relative to the output of bayesMixModel, e.g., any thinning specified here multiplies by the thinning that was specified in bayesMixModel.

cols	Number of columns to be used in the plot. Optional, if omitted, the number of columns and rows are choosen be the method itself.
...	Further arguments passed to plot.

Details

The number of iterations necessary until a Markov chain reaches stationarity depends on the specific model and data. For any inference based on Markov Chain Monte Carlo methods, it is therefore necessary to inspect the convergence of Markov Chains. One way to do this is visual inspection of trace plots using this method.

If argument main is passed to this method, it should have as many elements as chains are plotted. Otherwise, vector main is reapted.

Author(s)

Hans-Ulrich Klein (h.klein@uni-muenster.de) Martin Schaefer (martin.schaefer@udo.edu)

See Also

bayesMixModel, MixModelBayes-class

Examples

```
z <- c(rnorm(1000, 0, 3), rnorm(1000, 0, 5), rexp(1000, 5), -rexp(1000, 5))
mm <- bayesMixModel(z, normNull=1:2, expNeg=3, expPos=4,
  sdNormNullInit=c(1, 2), rateExpNegInit=8, rateExpPosInit=8,
  shapeNorm0=c(1, 1), scaleNorm0=c(1, 1),
  shapeExpNeg0=c(1, 1), scaleExpNeg0=c(1, 1),
  shapeExpPos0=c(1, 1), scaleExpPos0=c(1, 1),
  sdAlpha=1, itb=200, nmc=1000, thin=10)
plotChains(mm, chain="pi")
plotChains(mm, component=c(2,3))
```

---

plotClassification *Plot classification obtained from a mixture model.*

Description

This method visualizes the assignment of data points to the mixture components of the given mixture model. The components are plotted on the y-axis and the data on the x-axis. Data points are plotted in the color of the respective mixture component.

Usage

```
plotClassification(object, method, ...)
```

Arguments

object	An object of MixModel-class.
method	Depending on the type of the mixture model (ML, Bayes), different approaches to obtain a classification are available. Also the default approach may vary.
...	Further arguments passed to plot.

Details

If method is given, it must be a valid option for method classification. E.g., if bayesMixModel was used to create the mixture model, valid options are "maxDens", "median" and "mode".

Arguments "col" and "pch" can be given to specify the color and the shape of the points plotted. Their length must equal to the number of components.

Author(s)

Hans-Ulrich Klein (h.klein@uni-muenster.de)

See Also

MixModel-class listClassificationMethods

Examples

```
z <- c(rnorm(100, 0, 10), rnorm(100, 0, 2), rexp(100, 1/2), -rexp(100, 1/2))
mm <- mlMixModel(z, normNull=1:2, expNeg=3, expPos=4,
   sdNormNullInit=c(1, 2), rateExpNegInit=c(1/2), rateExpPosInit=c(1/2),
   pi=c(1/4, 1/4, 1/4, 1/4), maxIter=50, tol=0.01)
plotClassification(mm)
```

---

plotComponents *Plots the mixture density together with the densities of all single components.*

Description

This function plots the mixture pdf, the estimated data pdf and the weighted pdfs of all components of the given mixture model. The plot is useful to assess the fit of the model.

Usage

```
plotComponents(object, density = FALSE, ...)
```

Arguments

object	A MixModel-class object to be plotted.
density	A logical indicating whether the data distribution should be plotted as histogram (FALSE) or as density (TRUE) estimated using kernel density estimation.
...	Further arguments passed to plot.

Details

If the argument "col" is given, the first color is used for the mixture pdf. The following colors (2 to n+1) are used for the n mixture components' pdfs. If density is set to TRUE, a further color (n+2) must be given, that is used for the data pdf. The same applies for the argument "lty", which can be given to specify the line type used to plot the densities.

Author(s)

Hans-Ulrich Klein (h.klein@uni-muenster.de)

See Also

MixModel-class

Examples

```
z <- c(rnorm(100, 0, 1), rnorm(100, 0, 2), rexp(100, 1/2), -rexp(100, 1/2))
mm <- mlMixModel(z, normNull=1:2, expNeg=3, expPos=4,
   sdNormNullInit=c(1, 2), rateExpNegInit=c(1/2), rateExpPosInit=c(1/2),
   pi=c(1/4, 1/4, 1/4, 1/4), maxIter=50, tol=0.01)
plotComponents(mm)
```

---

summarizeReads *Count reads lying within given regions.*

Description

This function takes reads from e.g. ChIP-seq experiments and regions, e.g. promoters of genes, and assigns the number of overlapping reads to that region. This method was written particularly with regard to histone ChIP-seq experiments. Some histone modifications mainly occur at transcriptional start sites and thus ChIP-seq values can be assigned to genes by counting the number of reads within genes' pomoter regions. However, some genes may have several transcript and hence several promoters. Different options for handling multiple promoters are implemented. This method is also useful when integrating microarray expression data and ChIP-seq data, since most array platforms are gene centric and have probes that measure several transcripts.

Usage

```
summarizeReads(object, regions, summarize)
```

Arguments

object	A GRangesList with one GRanges object for each sample storing the ChIP-seq reads. The names of the GRangesList elements are used as sample names.
regions	An object of type GRangesList storing the promoter regions. Each element can be interpreted as gene or probe that has one or more promoters. The names of the lists' elements are used as row names. Alternatively, regions can be a GRanges object which as then handled like a GRangesList object with only one region in each list element. names of the GRanges are used as row names in this case.
summarize	Defines how regions with several ranges are handled. "average" means that the mean count of reads across all ranges is assigned to the region whereas "add" means that all counts are simply added (default).

Details

This function is usually applied after calling matchProbeToPromoter. When matchProbeToPromoter is used with mode "union", it is recommended to use "add". If the option "keepAll" had been used, one might want to use "average".

This method uses countOverlaps and counts each read that overlaps with at least one base.

Value

Returns a ChIPseqSet with number of rows equal to the length of regions and number of samples equal to the length of object.

Author(s)

Hans-Ulrich Klein (h.klein@uni-muenster.de)

See Also

matchProbeToPromoter ChIPseqSet-class

Examples

```
chipSeq <- GRangesList()
chipSeq[[1]] <- GRanges(seqnames=c("1", "1", "1", "1"),
    ranges=IRanges(start=c(97900, 198200, 198600, 202500),
        end=c(98100, 198400, 198800, 202700)),
    strand=c("+", "+", "+", "+"))
chipSeq[[2]] <- GRanges(seqnames=c("1", "1", "1", "1"),
    ranges=IRanges(start=c(97900, 198200, 198600, 300000),
        end=c(98100, 198400, 198800, 300200)),
    strand=c("+", "+", "+", "+"))
names(chipSeq) = c("sample1", "sample2")

promoters <- GRanges(seqnames=c("1", "1", "1"),
    ranges=IRanges(start=c(98000, 198000, 202000),
        end=c(101999, 201999, 205999)),
    strand=c("-", "+", "+"),
    probe=c("101", "102", "102"))
promoters <- split(promoters, elementMetadata(promoters)$probe)

chipSet <- summarizeReads(chipSeq, promoters, summarize="add")
chipVals(chipSet)
```

---

transToTSS *A data frame with Ensemble transcript IDs and transcriptional start sites.*

Description

The data frame stores Ensemble transcript IDs and repective chromosomes, transcriptional start sites and strands for mus musculus (mm10).

Usage

```
data(transToTSS)
```

Format

A data frame with 277 mouse transcripts with the following 4 variables:

`ensembl_transcript_id` A character giving the Ensemble transcript ID.

`chromosome_name` A character with the respective chromomse name.

`transcript_start` An integer storing the respective transcriptional start site.

`strand` An integer storing the respective strand information.

Details

Given a character vector `transcripts` with the Ensemble transcript IDs, a data frame like this can be obtained via biomaRt:

```
library("biomaRt") mart <-useMart("ensembl",dataset="mmusculus_gene_ensembl") transToTSS
<-getBM(attributes=c("ensembl_transcript_id","chromosome_name","transcript_start","transcript_en
```

Source

http://www.ensembl.org

Examples

```
data(transToTSS)
head(transToTSS)
```

Index