Package ‘BitSeq’

March 25, 2013

Type Package

Depends Rsamtools, zlibbioc

Imports IRanges

LinkingTo Rsamtools, zlibbioc

Title Transcript expression inference and differential expression analysis for RNA-seq data

Version 1.2.2

Date 2012-11-06

Author Peter Glaus, Antti Honkela and Magnus Rattray

Maintainer Peter Glaus <glaus@cs.man.ac.uk>

Description The BitSeq package is targeted for transcript expression analysis and differential expression analysis of RNA-seq data in two stage process. In the first stage it uses Bayesian inference methodology to infer expression of individual transcripts from individual RNA-seq experiments. The second stage of BitSeq embraces the differential expression analysis of transcript expression. Providing expression estimates from replicates of multiple conditions, Log-Normal model of the estimates is used for inferring the condition mean transcript expression and ranking the transcripts based on the likelihood of differential expression.

biocViews GeneExpression, DifferentialExpression,HighThroughputSequencing, RNAseq

R topics documented:

BitSeq-package *Bayesian Inference of Transcripts from Sequencing data*

Description

The BitSeq package is targeted for transcript expression analysis and differential expression analysis of RNA-seq data in two stage process. In the first stage it uses Bayesian inference methodology to infer expression of individual transcripts from individual RNA-seq experiments. The second stage of BitSeq embraces the differential expression analysis of transcript expression. Providing expression estimates from replicates of multiple conditions, Log-Normal model of the estimates is used for inferring the condition mean transcript expression and ranking the transcripts based on the likelihood of differential expression.

Details

Package:	BitSeq
Type:	Package
Version:	0.3.0
Date:	2012-03-09
License:	Artistic-2.0 + other

For details of using the package please refer to the Vignette.

Author(s)

Peter Glaus, Antti Honkela and Magnus Rattray Maintainer: Peter Glaus <glaus@cs.man.ac.uk>

References

Glaus, P., Honkela, A. and Rattray, M. (2012). Identifying differentially expressed transcripts from RNA-seq data with biological variation. Bioinformatics, 28(13), 1721-1728.

Examples

```
## Not run:
## basic use
res1 <- getExpression("data-c0b0.sam","ensSelect1.fasta")
res2 <- getExpression("data-c0b1.sam","ensSelect1.fasta")
res3 <- getExpression("data-c1b0.sam","ensSelect1.fasta")
res4 <- getExpression("data-c1b1.sam","ensSelect1.fasta")

deRes <- getDE( list(c(res1$fn, res2$fn),
              c(res3$fn, res4$fn)) )
## top 10 differentially expressed
head(deRes$pplr[ order(abs(0.5-deRes$pplr$pplr), decreasing=TRUE ), ], 10)

## advanced use, keeping the intermediate files
parseAlignment( "data-c0b0.sam",
  outFile = "data-c0b0.prob",
  trSeqFile = "ensSelect1.fasta",
  trInfoFile = "data.tr",
```

```
    uniform = TRUE,
    verbose = TRUE )

  estimateExpression( "data-c0b0.prob",
    outFile = "data-c0b0",
    outputType = "RPKM",
    trInfoFile = "data.tr",
    MCMC_burnIn = 200,
    MCMC_samplesN = 200,
    MCMC_samplesSave = 100,
    MCMC_scaleReduction = 1.1,
    MCMC_chainsN = 2 )

  cond1Files = c("data-c0b0.rpkm","data-c0b1.rpkm")
  cond2Files = c("data-c1b1.rpkm","data-c1b1.rpkm")
  allConditions = list(cond1Files, cond2Files)

  getMeanVariance( allConditions,
    outFile = "data.means",
    log = TRUE )

  estimateHyperPar( allConditions,
    outFile = "data.par",
    meanFile = "data.means",
    verbose = TRUE )

  estimateDE( allConditions,
    outFile = "data",
    parFile = "data.par" )


  ## End(Not run)
```

---

estimateDE — *Estimate condition mean expression and calculate Probability of Positive Log Ratio(PPLR)*

Description

Estimate condition mean expression for both experimental conditions using the expression estimates obtained by estimateExpression

Usage

```
estimateDE(conditions, outFile, parFile,
    lambda0=NULL, samples=NULL, confidencePerc=NULL,
    verbose=NULL, norm=NULL, pretend=FALSE )
```

Arguments

conditions	List of vectors, each vector containing names of files containing the expression samples from a replicate (Can be both technical and biological replicates. However, in order to get good results biological replicates for each condition are essential).

outFile	Prefix for the output files.
parFile	File containing estimated hyperparameters.
samples	Produce samples of condition mean expression apart from PPLR and confidence.
confidencePerc	Percentage for confidence intervals.
verbose	Verbose output. Advanced options:
lambda0	Model parameter lambda_0.
norm	Vector of (multiplicative) normalization constants for library size normalization of expression samples. Number of constants has to match the number of expression samples files.
pretend	Do not execute, only print out command line calls for the C++ version of the program.

Details

This function takes as an input expression samples from biological replicates of two or more conditions and hyperparameters over precision distribution inferred by estimateHyperPar. It uses pseudo-vectors of expression samples from all replicates to infer condition mean expression for each condition. The condition mean expression samples are used for computation of the Probability of Positive Log Ratio (PPLR) as well as $\log_2$ fold change of expression with confidence intervals and average condition mean expression for each transcript. Optionally the function can produce also the samples of condition mean expression for each condition.

Value

.pplr	file containing the PPLR, confidence interval, mean log2 fold change, mean condition mean expressions
.est	files containing samples of condition mean expressions for each condition - optional
.estVar	file containing samples of inferred variance of the first condition - optional

Author(s)

Peter Glaus

See Also

estimateExpression, estimateHyperPar

Examples

```
## Not run:
cond1Files = c("data-c0b0.rpkm","data-c0b1.rpkm")
cond2Files = c("data-c1b0.rpkm","data-c1b1.rpkm")
estimateDE(conditions=list(cond1Files, cond2Files), outFile="data.pplr", parFile="data.par",
        norm=c(1.0, 0.999, 1.0017, 0.9998))

## End(Not run)
```

estimateExpression *Estimate expression of transcripts*

Description

Estimates the expression of transcripts using Markov chain Monte Carlo Algorithm

Usage

```
estimateExpression(probFile, outFile, parFile=NULL, outputType=NULL, gibbs=NULL,
    trInfoFile=NULL, thetaActFile=NULL, MCMC_burnIn=NULL, MCMC_samplesN=NULL,
    MCMC_samplesSave=NULL, MCMC_chainsN=NULL, MCMC_dirAlpha=NULL, seed=NULL,
      verbose=NULL, pretend=FALSE)
estimateExpressionLegacy(probFile, outFile, parFile=NULL, outputType=NULL, gibbs=NULL,
    trInfoFile=NULL, thetaActFile=NULL, MCMC_burnIn=NULL, MCMC_samplesN=NULL,
    MCMC_samplesSave=NULL, MCMC_samplesNmax=NULL, MCMC_chainsN=NULL,
    MCMC_scaleReduction=NULL, MCMC_dirAlpha=NULL, seed=NULL, verbose=NULL,
      pretend=FALSE)
```

Arguments

probFile	File with alignment probabilities produced by parseAlignment
outFile	Prefix for the output files.
outputType	Output type, possible values: theta, RPKM, counts, tau.
gibbs	Use regular Gibbs sampling instead of Collapsed Gibbs sampling.
parFile	File containing parameters for the sampler, which can be otherwise specified by [MCMC*] options. As the file is checked after every MCMC iteration, the parameters can be adjusted while running.
trInfoFile	File containing transcript information. (Necessary for RPKM)
MCMC_burnIn	Length of sampler's burn in period.
MCMC_samplesN	Initial number of samples produced. These are used either to estimate the number of necessary samples or to estimate possible scale reduction.
MCMC_samplesSave	Number of samples recorder at the end in total.
MCMC_chainsN	Number of parallel chains used. At least two chains will be used.
seed	Sets the initial random seed for repeatable experiments.
verbose	Verbose output. Advanced options:
thetaActFile	File for logging noise parameter thetaAct, which is only generated when regular Gibbs sampling is used.
MCMC_dirAlpha	Alpha parameter for the Dirichlet distribution.
pretend	Do not execute, only print out command line calls for the C++ version of the program.

MCMC_scaleReduction
: (Only for estimateExpressionLegacy.) Target scale reduction, sampler finishes after this value is met.

MCMC_samplesNmax
: (Only for estimateExpressionLegacy.) Maximum number of samples produced in one iteration. After producing samplesNmax samples sampler finishes.

Details

This function runs Collapse Gibbs algorithm to sample the MCMC samples of transcript expression. The input is the .prob file containing alignment probabilities which were produced by parseAlignment. Other optional input is the transcript information file specified by trInfoFile and again produced by parseAlignment.

The estimateExpression function first runs burn-in phase and initial iterations to estimate the properties of the MCMC sampling. The initial samples are used to estimate the number of samples necessary for generating MCMC_samplesSave effective samples in the second, final, stage.

The estimateExpressionLegacy uses less efficient convergence checking via "scale reduction" estimation. After an iteration of generating MCMC_samplesN samples, it estimates possible scale reduction of the marginal posterior variance. While the possible scale reduction is high, it doubles the MCMC_samplesN and starts new iteration. This process is repeated until desired value of MCMC_scaleReduction is met, or MCMC_samplesNmax samples are generated.

The sampling algorithm can be configured via parameters file parFile or by using the MCMC* options. The advantage of using the file (at least an existing blank text document) is that by changing the configuration values while running, the new values do get updated after every iteration.

Value

.thetaMeans
: file containing average relative expression of transcripts θ

Either one of sample files based on output type selected:

.rpkm
: for RPKM expression

.counts
: for estimated read counts

.theta
: for relative expression of fragments

.tau
: for relative expression of transcripts

Author(s)

Peter Glaus

See Also

parseAlignment

Examples

```
## Not run:
estimateExpression( probFile="data.prob", outFile="data", outputType="RPKM",
    trInfoFile="data.tr", seed=47, verbose=TRUE)
estimateExpression( probFile="data-c0b0.prob", outFile="data-c0b0", outputType="RPKM",
  trInfoFile="data.tr", MCMC_burnIn=200, MCMC_samplesN=200, MCMC_samplesSave=100,
    MCMC_chainsN=2 , MCMC_dirAlpha=NULL )
estimateExpression( probFile="data.prob", outFile="data-G", gibbs=TRUE,
```

```
    parFile="parameters1.txt", outputType="counts", trInfoFile="data.tr")
estimateExpressionLegacy( probFile="data-c0b0.prob", outFile="data-c0b0", outputType="RPKM",
   trInfoFile="data.tr", MCMC_burnIn=200, MCMC_samplesN=200, MCMC_samplesSave=100,
   MCMC_samplesNmax=10000, MCMC_scaleReduction=1.2, MCMC_chainsN=2 , MCMC_dirAlpha=NULL )

## End(Not run)
```

---

estimateHyperPar *Estimate hyperparameters for DE model using expression samples and joint mean expression*

Description

Estimate hyperparameters for the Differential Expression model using expression samples and produced smoothed values of the hyperparameters depending on joint mean expression.

Usage

```
estimateHyperPar( outFile, conditions=NULL, paramsInFile=NULL,
      meanFile=NULL, force=TRUE, exThreshold=NULL, lambda0=NULL,
    paramsAllFile=NULL, smoothOnly=NULL, lowess_f=NULL, lowess_steps=NULL, verbose=NULL,
      veryVerbose=NULL, norm=NULL, pretend=FALSE )
```

Arguments

outFile	Name of the output file.
conditions	List of vectors, each vector containing names of files containing the expression samples from a replicate (Can be both technical and biological replicates. However, in order to get good results biological replicates for each condition are essential).
paramsInFile	File produced by previous run of the function using paramsAllFile flag.
meanFile	Name of the file containing joint mean and variance.
exThreshold	Threshold of lowest expression for which the estimation is done.
paramsAllFile	Name of the file to which to store all parameter values generated prior to lowess smoothing(good for later, more careful re-smoothing.)
smoothOnly	Input file contains previously sampled hyperparameters which should smoothed only.
verbose	Verbose output. Advanced options:
force	Force smoothing hyperparameters, otherwise program might not produce parameters file at the end.
lambda0	Model parameter lambda0.
lowess_f	Parameter F for lowess smoothing specifying amount of smoothing.
lowess_steps	Parameter Nsteps for lowess smoothing specifying number of iterations.
veryVerbose	More verbose output.
norm	Vector of (multiplicative) normalization constants for library size normalization of expression samples. Number of constants has to match the number of expression samples files.
pretend	Do not execute, only print out command line calls for the C++ version of the program.

Value

.par	file containing the smoothed hyperparameters
.ALLpar	file containing all hyperparameter samples prior to smoothing - optional

Author(s)

Peter Glaus

See Also

estimateDE

Examples

```
## Not run:
cond1Files = c("data-c0b0.rpkm","data-c0b1.rpkm")
cond2Files = c("data-c1b0.rpkm","data-c1b1.rpkm")
estimateHyperPar( conditions=list(cond1Files, cond2Files), outFile="data.par",
    meanFile="data.means", verbose=TRUE)

estimateHyperPar( conditions=list(cond1Files, cond2Files), outFile="data.par",
    meanFile="data.means", paramsFile="data.ALLpar", force=FALSE)
estimateHyperPar( outFile="data.par", paramsInFile="data.ALLpar", smoothOnly=TRUE )

## End(Not run)
```

---

getDE — *Estimate Probability of Positive Log Ratio*

Description

Using expression samples, program estimates the probability of differential expression for each transcript.

Usage

```
getDE(conditions, outPrefix=NULL, samples=FALSE, trInfoFile=NULL,
    norm=NULL, pretend=FALSE )
```

Arguments

conditions	List of vectors, each vector containing names of files containing the expression samples from a replicate (Can be both technical and biological replicates. However, in order to get good results biological replicates for each condition are essential).
outPrefix	Prefix for the output files. Otherwise program creates temporary files, which are only valid for current R session.
samples	Produce samples of condition mean expression apart from PPLR and confidence.
trInfoFile	Transcript information file providing the names of transcripts.

norm	Vector of (multiplicative) normalization constants for library size normalization of expression samples. Number of constants has to match the number of expression samples files.
pretend	Do not execute, only print out command line calls for the C++ version of the program.

Details

This function uses estimateHyperPar function to estimate the hyperparameters for DE model and the uses estimateDE function to infer the condition mean expression and calculate Probability of Positive Log Ratio.

Value

list with items:

pplr	DataFrame with PPLR and other statistics
fn	list with file names for PPLR file, fn$pplr, and condition mean expression samples, fn$samplesFiles (only with option samples=TRUE)

Author(s)

Peter Glaus

See Also

getExpression, estimateHyperPar, estimateDE

Examples

```
## Not run:
cond1Files = c("data-c0b0.rpkm","data-c0b1.rpkm")
cond2Files = c("data-c1b0.rpkm","data-c1b1.rpkm")
deRes <- getDE( conditions=list(cond1Files, cond2Files))
## top 10 DE transcripts
head(deRes$pplr[ order(abs(0.5-deRes$pplr$pplr), decreasing=TRUE ), ], 10)

## End(Not run)
```

---

getExpression *Estimate transcript expression*

Description

Estimate expression of transcripts. Starting from alignment and reference files function function handles the entire process of expression analysis resulting in transcript expression means and standard deviation together with file containing all the expression samples.

Usage

```
getExpression(alignFile, trSeqFile, outPrefix=NULL, uniform=TRUE, type="RPKM",
      log=FALSE, pretend=FALSE, ... )
```

Arguments

alignFile	File containing read alignments.
trSeqFile	File containing transcript sequence in FASTA format.
outPrefix	Prefix for the output files. Otherwise program creates temporary files, which are only valid for current R session.
uniform	Use uniform read distribution.
type	Output type, possible values: theta, RPKM, counts, tau.
log	Report mean and expression of logged expression samples.
pretend	Do not execute, only print out command line calls for the C++ version of the program.
...	Other arguments are passed to estimateExpression, please see estimateExpressionfor more details

Details

This function uses parseAlignment function to compute alignment probabilities and the function estimateExpression to produce the expression samples.

In case of non-uniform read distribution, it first produces approximate estimates of expression using uniform distribution and uses these estimates in to compute read distribution bias-corrected alignment probabilities, which are used in the estimateExpression function to produce expression estimates.

Value

list with items:

exp	DataFrame with transcript expression mean and standard deviation
fn	name of the file containing all the expression samples

Author(s)

Peter Glaus

See Also

getDE, estimateExpression, parseAlignment

Examples

```
## Not run:
res1 <- getExpression("data-c0b0.sam","ensSelect1.fasta", MCMC_chains=2,
    MCMC_samplesN=100)

## End(Not run)
```

getGeneExpression *Calculate gene expression or relative within gene expression*

Description

Calculate either gene expression or relative within gene expression using transcript expression samples and transcript information file.

Usage

```
getGeneExpression(sampleFile, outFile=NULL, trInfoFile=NULL, pretend=FALSE)
getWithinGeneExpression(sampleFile, outFile=NULL, trInfoFile=NULL, pretend=FALSE)
```

Arguments

sampleFile	File containing the transcript expression samples.
outFile	Name of the output file. If not used, function uses temporary file.
trInfoFile	Transcript information file. If not used, function tries file with same name and extension tr. The file has to contain valid gene transcript mapping, see detail below.
pretend	Do not execute, only print out command line calls for the C++ version of the program.

Details

The getGeneExpression function takes samples of transcript expression and produces file with expression of genes by adding up transcript expression.

The getWithinGeneExpression function takes samples of transcript expression and produces file with relative within gene expression samples for each transcript.

Both function need valid transcript information file which contains gene transcript mapping. The first line should contain "# M <numberOfTranscripts>" and the following numberOfTranscripts lines have to contain "<geneName> <transcriptName> <transcriptLength>". Example is provided in extdata/ensSelect1.tr. Please note that the transcript information file automatically generated from alignment files are not sufficient because SAM/BAM files do not include gene names. We hope to provide more convenient way in future versions of BitSeq.

Value

Name of file containing the new expression samples.

Author(s)

Peter Glaus

See Also

getExpression

Examples

```
setwd(system.file("extdata",package="BitSeq"))
## gene expression
getGeneExpression("data-c0b1.rpkm", "data-c0b1-GE.rpkm", "ensSelect1.tr")
gExpSamples <- loadSamples("data-c0b1-GE.rpkm")
gExpMeans <- rowMeans(as.data.frame(gExpSamples))
gExpMeans

## within gene expression
wgeFN <- getWithinGeneExpression("data-c0b1.rpkm", trInfoFile="ensSelect1.tr")
wgExpSamples <- loadSamples(wgeFN)
wgExpMeans <- rowMeans(as.data.frame(wgExpSamples))
head(wgExpMeans)
```

---

getMeanVariance *Calculate mean and variance of expression samples*

Description

Calculate mean and variance of expression samples or log-expression samples

Usage

```
getMeanVariance(sampleFiles, outFile, log=NULL, type=NULL, verbose=NULL,
      norm=NULL, pretend=FALSE)
```

Arguments

sampleFiles	Vector of one or more files containing the expression samples.
outFile	Name of the output file.
log	Use logged values.
type	Type of variance, possible values: sample,sqDif for sample variance or squared difference.
verbose	Verbose output.
norm	Vector of (multiplicative) normalization constants for library size normalization of expression samples. Number of constants has to match the number of expression samples files.
pretend	Do not execute, only print out command line calls for the C++ version of the program.

Details

The getMeanVariance function computes means and variances of MCMC expression samples. These can be computed either from single file or from multiple files using sample variance. Variance of two experiments (i.e. technical or biological replicates) can be estimated also by using sqDif option for type which specify the computation of the average square distance between the samples from two sets.

Value

.means	File containing means (first column) and variance (second column) for each transcript (or row in the sample files)

Author(s)

Peter Glaus

See Also

estimateExpression

Examples

```
setwd(system.file("extdata",package="BitSeq"));
sampleFileNames = c("data-c1b0.rpkm","data-c1b1.rpkm")
getMeanVariance(sampleFiles=sampleFileNames, outFile="data-c1.Lmean", log=1, norm=c(1.0017, 0.9998))
```

---

loadSamples *Loading and saving expression samples*

Description

Functions for loading expression samples into DataFrame and saving samples from DataFrame into a file.

Usage

```
loadSamples(fileName, trInfoFile=NULL)
writeSamples(data, fileName)
```

Arguments

fileName	Name of the file with samples or to which the samples are written.
data	DataFrame with samples written to the file.
trInfoFile	Transcript information file which can be used to name the rows.

Details

The loadSamples function load samples from the specified file into a DataFrame. If the transcript information file is provided, the transcript names are use as row names.

The writeSamples function can save samples from a DataFrame into a file in format which is valid for BitSeq and can be used in other functions.

Value

DataFrame	Containing the expression samples

Author(s)

Peter Glaus

See Also

estimateExpression

Examples

```
## Not run:
samples1<-loadSamples("data-c0b1.rpkm")
writeSamples(samples1,"new-c0b1.rpkm")

## End(Not run)
```

---

parseAlignment — *Compute probabilities of alignments*

Description

Compute probability of alignments and save them into *.prob* file.

Usage

```
parseAlignment( alignFile, outFile, trSeqFile, inputFormat=NULL, trInfoFile=NULL,
    expressionFile=NULL, readsN=NULL, uniform=TRUE, lenMu=NULL, lenSigma=NULL,
      verbose=NULL, veryVerbose=NULL, pretend=FALSE)
```

Arguments

alignFile	File containing read alignments.
outFile	Name of the output file.
inputFormat	Input format: possible values SAM, BAM.
trInfoFile	If transcript reference sequence information is contained within SAM file, program will write this information into <trInfoFile>, otherwise it will look for this information in the <trInfoFile>.
trSeqFile	File containing transcript sequence in FASTA format.
expressionFile	Transcript relative expression estimates — for better non-uniform read distribution estimation.
readsN	Total number of reads. This is usually not necessary if SAM/BAM contains also reads with no valid alignments.
uniform	Use uniform read distribution.
lenMu	Set mean of log fragment length distribution. $l_{frag} \sim LogNormal(\mu, \sigma^2)$
lenSigma	Set σ^2 (or variance) of log fragment length distribution. $l_{frag} \sim LogNormal(\mu, \sigma^2)$
verbose	Verbose output.
veryVerbose	Very verbose output.
pretend	Do not execute, only print out command line calls for the C++ version of the program.

Details

This function uses the alignments and reference file to assign probability to each alignment. It uses either bias-corrected or uniform model for the read distribution, assumes Log-Normal distribution of fragment lengths for pair-end read data and uses quality scores and mismatches to assign probability for every alignment of a read (or fragment) to a transcript.

Value

.prob	file containing the alignment probabilities
.tr	file containing reference transcript names, lengths and effective lengths - optional

Author(s)

Peter Glaus

See Also

estimateExpression

Examples

```
## Not run:
parseAlignment(alignFile="data.sam", outFile="data.prob",
    trSeqFile="trReference.fa" ,trInfoFile="data.tr")

## End(Not run)
```

Index