Package 'RCAS'

April 12, 2018

Type Package

Title RNA Centric Annotation System

Version 1.4.4

Date 2018-03-23

Description RCAS is an automated system that provides dynamic genome annotations for custom input files that contain transcriptomic regions. Such transcriptomic regions could be, for instance, peak regions detected by CLIP-Seq analysis that detect protein-RNA interactions, RNA modifications (alias the epitranscriptome), CAGE-tag locations, or any other collection of target regions at the level of the transcriptome. RCAS is designed as a reporting tool for the functional analysis of RNA-binding sites detected by high-throughput experiments. It takes as input a BED format file containing the genomic coordinates of the RNA binding sites and a GTF file that contains the genomic annotation features usually provided by publicly available databases such as Ensembl and UCSC. RCAS performs overlap operations between the genomic coordinates of the RNA binding sites and the genomic annotation features and produces in-depth annotation summaries such as the distribution of binding sites with respect to gene features (exons, introns, 5'/3' UTR regions, exon-intron boundaries, promoter regions, and whole transcripts). Moreover, by detecting the collection of targeted transcripts, RCAS can carry out functional annotation tables for enriched gene sets (annotated by the Molecular Signatures Database) and GO terms. As one of the most important questions that arise during protein-RNA interaction analysis; RCAS has a module for detecting sequence motifs enriched in the targeted regions of the transcriptome. A full interactive report in HTML format can be generated that contains interactive figures and tables that are ready for publication purposes.

License Artistic-2.0

LazyData TRUE

Depends R (>= 3.3.0), plotly (>= 4.5.2), DT (>= 0.2), data.table, topGO, motifRG

Imports biomaRt, AnnotationDbi, GenomicRanges, BSgenome.Hsapiens.UCSC.hg19, GenomeInfoDb (>= 1.12.0), Biostrings, rtracklayer, org.Hs.eg.db, GenomicFeatures, rmarkdown (>= 0.9.5), genomation (>= 1.5.5), knitr (>= 1.12.3), BiocGenerics, S4Vectors, stats, plotrix, pbapply, RSQLite, proxy, DBI, pheatmap, ggplot2, cowplot, ggseqlogo, methods,

utils

RoxygenNote 6.0.1

Suggests BSgenome.Hsapiens.UCSC.hg38, BSgenome.Mmusculus.UCSC.mm10, BSgenome.Mmusculus.UCSC.mm9, BSgenome.Celegans.UCSC.ce10, BSgenome.Dmelanogaster.UCSC.dm3, org.Mm.eg.db, org.Ce.eg.db, org.Dm.eg.db, testthat, covr

SystemRequirements pandoc (>= 1.12.3)

VignetteBuilder knitr

biocViews Software, GeneTarget, MotifAnnotation, MotifDiscovery, GO, Transcriptomics, GenomeAnnotation, GeneSetEnrichment, Coverage

NeedsCompilation no

Author Bora Uyar [aut, cre], Dilmurat Yusuf [aut], Ricardo Wurmus [aut], Altuna Akalin [aut]

Maintainer Bora Uyar <bora.uyar@mdc-berlin.de>

R topics documented:

`calculateCoverageProfile` *calculateCoverageProfile*

Description

This function checks overlaps between input query regions and annotation features, and then calculates coverage profile along target regions.

Usage

```
calculateCoverageProfile(queryRegions, targetRegions, sampleN = 0,
  bin.num = 100, bin.op = "mean", strand.aware = TRUE)
```

Arguments

`queryRegions`	GRanges object imported from a BED file using `importBed` function
`targetRegions`	GRanges object containing genomic coordinates of a target feature (e.g. exons)
`sampleN`	If set to a positive integer, `targetRegions` will be downsampled to `sampleN` regions
`bin.num`	Positive integer value (default: 100) to determine how many bins the targetRegions should be split into (See genomation::ScoreMatrixBin)
`bin.op`	The operation to apply for each bin: 'min', 'max', or 'mean' (default: mean). (See genomation::ScoreMatrixBin)
`strand.aware`	TRUE/FALSE (default: TRUE) The strands of target regions are considered.

Value

A ScoreMatrix object returned by `genomation::ScoreMatrixBin` function. Target regions are divided into 100 equal sized bins and coverage level is calculated in a strand-specific manner.

Examples

```
data(gff)
data(queryRegions)
txdbFeatures <- getTxdbFeaturesFromGRanges(gffData = gff)
df <- calculateCoverageProfile(queryRegions = queryRegions,
                               targetRegions = txdbFeatures$exons,
                                     sampleN = 1000)
```

calculateCoverageProfileFromTxdb

calculateCoverageProfileFromTxdb

Description

This function overlaps the input query regions with a target list of annotation features and calculates the coverage profile along the target regions.

Usage

```
calculateCoverageProfileFromTxdb(queryRegions, txdb, type, sampleN = 0)
```

Arguments

queryRegions	GRanges object imported from a BED file using `importBed` function
txdb	A txdb object obtained by using `GenomicFeatures::makeTxDb` family of functions
type	A character string defining the type of gene feature for which a profile should be calculated. The options are: transcripts, exons, introns, promoters, fiveUTRs, threeUTRs, and cds.
sampleN	If set to a positive integer, the targetRegions will be downsampled to `sampleN` regions

Value

A data.frame object consisting of two columns: 1. coverage level 2. bins. The target regions are divided into 100 equal sized bins and coverage level is summarized in a strand-specific manner using the `genomation::ScoreMatrixBin` function.

Examples

```
data(gff)
data(queryRegions)
txdb <- GenomicFeatures::makeTxDbFromGRanges(gff)
df <- calculateCoverageProfileFromTxdb(queryRegions = queryRegions,
                                       type = 'exons',
                                       txdb = txdb,
                                    sampleN = 1000)
```

calculateCoverageProfileList

calculateCoverageProfileList

Description

This function checks overlaps between input query regions and a target list of annotation features, and then calculates the coverage profile along the target regions.

Usage

```
calculateCoverageProfileList(queryRegions, targetRegionsList, sampleN = 0,
  bin.num = 100, bin.op = "mean", strand.aware = TRUE)
```

Arguments

queryRegions	GRanges object imported from a BED file using importBed function
targetRegionsList	A list of GRanges objects containing genomic coordinates of target features (e.g. transcripts, exons, introns)
sampleN	If set to a positive integer, targetRegions will be downsampled to sampleN regions
bin.num	Positive integer value (default: 100) to determine how many bins the targetRegions should be split into (See genomation::ScoreMatrixBin)
bin.op	The operation to apply for each bin: 'min', 'max', or 'mean' (default: mean). (See genomation::ScoreMatrixBin)
strand.aware	TRUE/FALSE (default: TRUE) The strands of target regions are considered.

Value

A data.frame consisting of four columns: 1. bins level 2. meanCoverage 3. standardError 4. feature Target regions are divided into 100 equal sized bins and coverage level is summarized in a strand-specific manner using the genomation::ScoreMatrixBin function. For each bin, mean coverage score and the standard error of the mean coverage score is calculated (plotrix::std.error)

Examples

```
data(gff)
data(queryRegions)
txdbFeatures <- getTxdbFeaturesFromGRanges(gffData = gff)
dfList <- calculateCoverageProfileList(queryRegions = queryRegions,
                               targetRegionsList = txdbFeatures,
                                     sampleN = 1000)
```

---

calculateCoverageProfileListFromTxdb

calculateCoverageProfileListFromTxdb

Description

This function overlaps the input query regions with a target list of annotation features and calculates the coverage profile along the target regions.

Usage

```
calculateCoverageProfileListFromTxdb(queryRegions, txdb, sampleN = 0)
```

Arguments

queryRegions	GRanges object imported from a BED file using importBed function
txdb	A txdb object obtained by using GenomicFeatures::makeTxDb family of functions
sampleN	If set to a positive integer, targetRegions will be downsampled to sampleN regions

Value

A list of data.frame objects consisting of two columns: 1. coverage level 2. bins. The target regions are divided into 100 equal sized bins and coverage level is summarized in a strand-specific manner using the genomation::ScoreMatrixBin function.

Examples

```
data(gff)
data(queryRegions)
txdb <- GenomicFeatures::makeTxDbFromGRanges(gff)
df <- calculateCoverageProfileListFromTxdb(queryRegions = queryRegions,
                                           txdb = txdb,
                                          sampleN = 1000)
```

---

createControlRegions *createControlRegions*

Description

Given a GRanges object of query regions, create a background set of peaks that have the same length distribution based on the flanking regions of the peaks.

Usage

```
createControlRegions(queryRegions)
```

Arguments

queryRegions	GRanges object containing coordinates of input query regions imported by the importBed function.

Value

GRanges object that contains the same number of regions as query regions

Examples

```
data(queryRegions)
controlRegions <- createControlRegions(queryRegions = queryRegions)
```

createDB *createDB*

Description

Creates an sqlite database consisting of various tables of data obtained from processed BED files

Usage

```
createDB(dbPath = file.path(getwd(), "rcasDB.sqlite"), projDataFile,
  gtfFilePath = "", update = FALSE, genomeVersion,
  annotationSummary = TRUE, coverageProfiles = TRUE, motifAnalysis = TRUE,
  nodeN = 1)
```

Arguments

dbPath	Path to the sqlite database file (could be an existing file or a new file path to be created at the given path)
projDataFile	A file consisting of meta-data about the input samples. Must minimally consist of two columns: 1. sampleName (name of the sample) 2. bedFilePath (full path to the location of the BED file containing data for the sample)
gtfFilePath	Path to the GTF file (preferably downloaded from the Ensembl database) that contains genome annotations
update	TRUE/FALSE (default: FALSE) whether an existing database should be updated
genomeVersion	A character string to denote for which genome version the analysis is being done. Available options are hg19/hg38 (human), mm9/mm10 (mouse), ce10 (worm) and dm3 (fly).
annotationSummary	TRUE/FALSE (default:TRUE) whether annotation summary module should be run
coverageProfiles	TRUE/FALSE (default: TRUE) whether coverage profiles module should be run
motifAnalysis	TRUE/FALSE (default: TRUE) whether motif discovery module should be run
nodeN	Number of cpus to use for parallel processing (default: 1)

Value

Path to an SQLiteConnection object created by RSQLite package

Examples

```
FUS_path <- system.file("extdata", "FUS_Nakaya2013c_hg19.bed",
package='RCAS')

FMR1_path <- system.file("extdata",
"FMR1_Ascano2012a_hg19.bed", package='RCAS')

projData <- data.frame('sampleName' = c('FUS', 'FMR1'),
'bedFilePath' = c(FUS_path,FMR1_path), stringsAsFactors = FALSE)
```

```
write.table(projData, 'myProjDataFile.tsv', sep = '\t', quote =FALSE,
row.names = FALSE)

gtfFilePath <- system.file("extdata",
"hg19.sample.gtf", package='RCAS')

createDB(dbPath = 'hg19.RCASDB.sqlite',
projDataFile = './myProjDataFile.tsv',
gtfFilePath = gtfFilePath,
genomeVersion = 'hg19',
motifAnalysis = FALSE,
coverageProfiles = FALSE)

#Note: to add new data to an existing database, set update = TRUE
```

`createOrthologousGeneSetList` *createOrthologousMsigdbDataset*

Description

Gene set annotations in public databases are usually geared towards human. This function is used to utilize human gene set annotations to create such gene sets for other species such as mouse, fly, and worm via orthologous relationships to human genes.

Usage

```
createOrthologousGeneSetList(referenceGeneSetList, refGenomeVersion = "hg19",
  targetGenomeVersion)
```

Arguments

`referenceGeneSetList`
: A named list of vectors where each vector consists of a set of Entrez gene ids (for instance, returned by `parseMsigdb` function

`refGenomeVersion`
: Genome version of a reference species. (default:hg19)

`targetGenomeVersion`
: Genome version of a target species. Available options are mm9, dm3, and ce10

Value

A list of vectors where each vector consists of a set of Entrez gene ids

Examples

```
#Recommended gene sets (with Entrez Ids) from MSIGDB database can be downloaded
#from \url{http://software.broadinstitute.org/gsea/msigdb/collections.jsp#C2}
#Here we use built-in random gene sets to show how the function works
data(geneSets)
#Map the gene sets to a target genome (supported genomes: mm9, dm3, or ce10)
```

```
orthGeneSets <- createOrthologousGeneSetList(
                              referenceGeneSetList = geneSets,
                              refGenomeVersion = 'hg19',
                              targetGenomeVersion = 'mm9'
                              )
```

`deleteSampleDataFromDB` *deleteSampleDataFromDB*

Description

Given a list of sample names, the function deletes all datasets calculated for the given samples from the database.

Usage

```
deleteSampleDataFromDB(dbPath, sampleNames)
```

Arguments

`dbPath`	Path to the sqlite database
`sampleNames`	The names of the samples for which all relevant datasets should be deleted from the database. Tip: Use RSQLite::dbReadTable function to read the table 'processedSamples' to see which samples are available in the database.

Value

SQLiteConnection object with updated contents in the dbPath

`discoverFeatureSpecificMotifs` *discoverFeatureSpecificMotifs*

Description

This function groups query regions based on their overlap with different transcript features and generates a table of top enriched motif and matching patterns for each given transcript feature type along with some other motif discovery related statistics.

Usage

```
discoverFeatureSpecificMotifs(queryRegions, txdbFeatures, ...)
```

Arguments

queryRegions	GRanges object containing coordinates of input query regions imported by the importBed function
txdbFeatures	A list of GRanges objects where each GRanges object corresponds to the genomic coordinates of gene features such as promoters, introns, exons, 5'/3' UTRs and whole transcripts. This list of GRanges objects are obtained by the function getTxdbFeaturesFromGRanges or getTxdbFeatures.
...	Other arguments passed to runMotifRG function. Important arguments are 'genomeVersion' and motifN. If motifN is bigger than 1, then multiple motifs will be found but only the top motif will be plotted.

Value

A data.frame object

Examples

```
## Not run:
data(gff)
data(queryRegions)
txdbFeatures <- getTxdbFeaturesFromGRanges(gffData = gff)
discoverFeatureSpecificMotifs(queryRegions = queryRegions,
genomeVersion = 'hg19', txdbFeatures = txdbFeatures,
motifN = 1, nCores = 1)
## End(Not run)
```

---

extractSequences *extractSequences*

Description

Given a GRanges object and a genome version (hg19, mm9, ce10 or dm3), this function extracts the DNA sequences for all genomic regions found in an input object.

Usage

```
extractSequences(queryRegions, genomeVersion)
```

Arguments

queryRegions	GRanges object containing coordinates of input query regions imported by the importBed function
genomeVersion	A character string to denote the BS genome library required to extract sequences. Available options are hg19, mm9, ce10 and dm3.

Value

DNAStringSet object will be returned

Examples

```
data(queryRegions)
sequences <- extractSequences(queryRegions = queryRegions,
                              genomeVersion = 'hg19')
```

`geneSets` *Random test gene sets*

Description

This dataset contains random sets of genes with Entrez ids that is designed to represent the data that can be parsed from MSIGDB database. using the `parseMsigdb` function.

Usage

```
geneSets
```

Format

A list of vectors, where each list element corresponds to a (randomized) gene set, where genes are represented by Entrez ids.

Details

Actual curated datasets must be downloaded from the MSIGDB database

Value

A list object

`getFeatureBoundaryCoverage` *getFeatureBoundaryCoverage*

Description

This function extracts the flanking regions of 5' and 3' boundaries of a given set of genomic features and computes the per-base coverage of query regions across these boundaries.

Usage

```
getFeatureBoundaryCoverage(queryRegions, featureCoords, flankSize = 500,
  boundaryType, sampleN = 0)
```

Arguments

queryRegions	GRanges object imported from a BED file using importBed function
featureCoords	GRanges object containing the target feature coordinates
flankSize	Positive integer that determines the number of base pairs to extract around a given genomic feature boundary
boundaryType	(Options: fiveprime or threeprime). Denotes which side of the feature's boundary is to be profiled.
sampleN	A positive integer value less than the total number of featuer coordinates that determines whether the target feature coordinates should be randomly downsampled. If set to 0, no downsampling will happen. If

Value

a data frame containin three columns. 1. fivePrime: Coverage at 5' end of features 2. threePrime: Coverage at 3' end of features; 3. bases: distance (in bp) to the boundary

Examples

```
data(queryRegions)
data(gff)
txdb <- GenomicFeatures::makeTxDbFromGRanges(gff)
transcriptCoords <- GenomicFeatures::transcripts(txdb)
transcriptEndCoverage <- getFeatureBoundaryCoverage (
                                     queryRegions = queryRegions,
                                    featureCoords = transcriptCoords,
                                    flankSize = 100,
                                    boundaryType = 'threeprime',
                                    sampleN = 1000)
```

---

getFeatureBoundaryCoverageBin

getFeatureBoundaryCoverageBin

Description

This function extracts the flanking regions of 5' and 3' boundaries of a given set of genomic features, splits them into 100 equally sized bins and computes the per-bin coverage of query regions across these boundaries.

Usage

```
getFeatureBoundaryCoverageBin(queryRegions, featureCoords, flankSize = 50,
  sampleN = 0)
```

Arguments

queryRegions	GRanges object imported from a BED file using importBed function
featureCoords	GRanges object containing the target feature coordinates
flankSize	Positive integer that determines the number of base pairs to extract around a given genomic feature boundary

sampleN — A positive integer value less than the total number of featuer coordinates that determines whether the target feature coordinates should be randomly down-sampled. If set to 0, no downsampling will happen. If

Value

a data frame containin three columns. 1. fivePrime: Coverage at 5' end of features 2. threePrime: Coverage at 3' end of features; 3. bases: distance (in bp) to the boundary

Examples

```
data(queryRegions)
data(gff)
txdb <- GenomicFeatures::makeTxDbFromGRanges(gff)
transcriptCoords <- GenomicFeatures::transcripts(txdb)
transcriptEndCoverageBin <- getFeatureBoundaryCoverageBin (
                                     queryRegions = queryRegions,
                                    featureCoords = transcriptCoords,
                                    flankSize = 100,
                                    sampleN = 1000)
```

---

getFeatureBoundaryCoverageMulti

getFeatureBoundaryCoverageMulti

Description

This function is a wrapper function to run RCAS::getFeatureBoundaryCoverage multiple times, which is useful to get coverage signals across different kinds of transcript features for a given list of bed files imported as a GRangesList object.

Usage

```
getFeatureBoundaryCoverageMulti(bedData, txdbFeatures, sampleN = 10000)
```

Arguments

bedData — GRangesList object imported from multiple BED files using `importBedFiles` function

txdbFeatures — List of GRanges objects - outputs of `getTxdbFeaturesFromGRanges` and `getTxdbFeatures` functions

sampleN — (default=10000) Positive integer value that is used to randomly down-sample the target feature coordinates to improve the runtime. Set to 0 to avoid down-sampling.

Value

A data.frame object with coverage data at three prime and five prime boundaries of a list of transcript features

Examples

```
data(gff)
data(queryRegions)
queryRegionsList <- GRangesList(queryRegions, queryRegions)
names(queryRegionsList) <- c('q1', 'q2')
txdbFeatures <- getTxdbFeaturesFromGRanges(gffData = gff)
getFeatureBoundaryCoverageMulti(queryRegionsList, txdbFeatures, sampleN = 500)
```

`getIntervalOverlapMatrix` *getIntervalOverlapMatrix*

Description

This function is used to obtain a binary matrix of overlaps between a list of GRanges objects (GRangesList object) and a target GRanges object. The resulting matrix has N rows where N is the number of intervals in the target GRanges object and M columns where M is the number GRanges objects in the query GRangesList object.

Usage

```
getIntervalOverlapMatrix(queryRegionsList, targetRegions,
  targetRegionNames = NULL, nodeN = 1)
```

Arguments

`queryRegionsList`	A GRangesList object
`targetRegions`	A GRanges object
`targetRegionNames`	Optional vector of names to be used as rownames in the resulting matrix. The vector indices must correspond to the intervals in targetRegions object.
`nodeN`	Positive integer value to use one or more cpus for parallel computation (default: 1)

Value

A binary matrix object consisting of number of rows equal to the number of intervals in targetRegions object, and number of columns equal to the number of GRanges objects available in the queryRegionsList object.

Examples

```
data(gff)
input1 <- system.file("extdata", "testfile.bed", package='RCAS')
input2 <- system.file("extdata", "testfile2.bed", package='RCAS')
bedData <- RCAS::importBedFiles(filePaths = c(input1, input2))
M <- RCAS::getIntervalOverlapMatrix(
queryRegionsList = bedData,
targetRegions = gff[gff$type == 'gene',][1:100],
targetRegionNames = gff[gff$type == 'gene',][1:100]$gene_name)
```

getMotifSummaryTable *getMotifSummaryTable*

Description

A repurposed/simplified version of the `motifRG::summaryMotif` function.

Usage

```
getMotifSummaryTable(motifResults)
```

Arguments

`motifResults`	Output object of `runMotifRG` function

Value

A data.frame object containing summary statistics about the discovered motifs

Examples

```
data(queryRegions)
motifResults <- runMotifRG(queryRegions = queryRegions,
                           genomeVersion = 'hg19',
                           resize = 15,
                           motifN = 1,
                           nCores = 2)
motifSummary <- getMotifSummaryTable(motifResults)
```

getTargetedGenesTable *getTargetedGenesTable*

Description

This function provides a list of genes which are targeted by query regions and their corresponding numbers from an input BED file. Then, the hits are categorized by the gene features such as promoters, introns, exons, 5'/3' UTRs and whole transcripts.

Usage

```
getTargetedGenesTable(queryRegions, txdbFeatures)
```

Arguments

`queryRegions`	GRanges object containing coordinates of input query regions imported by the `importBed` function
`txdbFeatures`	A list of GRanges objects where each GRanges object corresponds to the genomic coordinates of gene features such as promoters, introns, exons, 5'/3' UTRs and whole transcripts. This list of GRanges objects are obtained by the function `getTxdbFeaturesFromGRanges` or `getTxdbFeatures`.

Value

A data.frame object where rows correspond to genes and columns correspond to gene features

Examples

```
data(gff)
data(queryRegions)
txdbFeatures <- getTxdbFeaturesFromGRanges(gffData = gff)
featuresTable <- getTargetedGenesTable(queryRegions = queryRegions,
                                       txdbFeatures = txdbFeatures)

#or
## Not run:
txdb <- GenomicFeatures::makeTxDbFromGRanges(gff)
txdbFeatures <- getTxdbFeatures(txdb)
featuresTable <- getTargetedGenesTable(queryRegions = queryRegions,
                                       txdbFeatures = txdbFeatures)

## End(Not run)
```

`getTxdbFeatures` *getTxdbFeatures*

Description

This function takes as input a txdb object from GenomicFeatures library. Then extracts the coordinates of gene features such as promoters, introns, exons, 5'/3' UTRs, and whole transcripts.

Usage

```
getTxdbFeatures(txdb)
```

Arguments

`txdb`	A txdb object imported by GenomicFeatures::makeTxDb family of functions

Value

A list of GRanges objects

Examples

```
data(gff)
txdb <- GenomicFeatures::makeTxDbFromGRanges(gff)
txdbFeatures <- getTxdbFeatures(txdb)
```

getTxdbFeaturesFromGRanges *getTxdbFeaturesFromGRanges*

Description

This function takes as input a GRanges object that contains GTF file contents (e.g from the output of `importGtf` function). Then extracts the coordinates of gene features such as promoters, introns, exons, 5'/3' UTRs and whole transcripts.

Usage

```
getTxdbFeaturesFromGRanges(gffData)
```

Arguments

`gffData`	A GRanges object imported by `importGtf` function

Value

A list of GRanges objects

Examples

```
data(gff)
txdbFeatures <- getTxdbFeaturesFromGRanges(gffData = gff)
```

gff *Sample GFF file imported as a GRanges object*

Description

This dataset contains genomic annotation data from Ensembl version 75 for Homo sapiens downloaded from Ensembl. The GFF file is imported via the `importGtf` function and a subset of the data is selected by choosing features found on 'chr1'.

Usage

```
gff
```

Format

GRanges object with 238010 ranges and 16 metadata columns

Value

A GRanges object

Source

ftp://ftp.ensembl.org/pub/release-75/gtf/homo_sapiens/Homo_sapiens.GRCh37.75.gtf.gz

---

`importBed` *importBed*

Description

This function uses `rtracklayer::import.bed()` function to import BED files

Usage

```
importBed(filePath, sampleN = 0, keepStandardChr = TRUE, debug = TRUE,
  ...)
```

Arguments

`filePath`	Path to a BED file
`sampleN`	A positive integer value. The number of intervals in the input BED file are randomly downsampled to include intervals as many as `sampleN`. The input will be downsampled only if this value is larger than zero and less than the total number of input intervals.
`keepStandardChr`	TRUE/FALSE (default:TRUE). If set to TRUE, will convert the `seqlevelsStyle` to 'UCSC' and apply `keepStandardChromosomes` function to only keep data from the standard chromosomes
`debug`	TRUE/FALSE (default:TRUE). Set to FALSE to turn off messages
`...`	Other arguments passed to rtracklayer::import.bed function

Value

A `GRanges` object containing the coordinates of the intervals from an input BED file

Examples

```
input <- system.file("extdata", "testfile.bed", package='RCAS')
importBed(filePath = input, keepStandardChr = TRUE)
```

importBedFiles *importBedFiles*

Description

This function is a wrapper that uses RCAS::importBed() function to import BED files as a GRangesList object

Usage

```
importBedFiles(filePaths, ...)
```

Arguments

filePaths	A vector of paths to one or more BED files
...	Other parameters passed to RCAS::importBed and rtracklayer::import.bed function

Value

A GRangesList object containing the coordinates of the intervals from multiple input BED files

Examples

```
input1 <- system.file("extdata", "testfile.bed", package='RCAS')
input2 <- system.file("extdata", "testfile2.bed", package='RCAS')
bedData <- importBedFiles(filePaths = c(input1, input2),
keepStandardChr = TRUE)
# when importing multiple bed files with different column names, it
# is required to pass the common column names to be parsed from the
# bed files
bedData <- importBedFiles(filePaths = c(input1, input2),
                colnames = c('chrom', 'start', 'end', 'strand'))
```

importGtf *importGtf*

Description

This function uses rtracklayer::import.gff() function to import genome annoatation data from an Ensembl gtf file

Usage

```
importGtf(filePath, saveObjectAsRds = TRUE, readFromRds = TRUE,
  overwriteObjectAsRds = FALSE, keepStandardChr = TRUE, ...)
```

Arguments

filePath	Path to a GTF file
saveObjectAsRds	TRUE/FALSE (default:TRUE). If it is set to TRUE, a GRanges object will be created and saved in RDS format (<filePath>.granges.rds) so that importing can re-use this .rds file in next run.
readFromRds	TRUE/FALSE (default:TRUE). If it is set to TRUE, annotation data will be imported from previously generated .rds file (<filePath>.granges.rds).
overwriteObjectAsRds	TRUE/FALSE (default:FALSE). If it is set to TRUE, existing .rds file (<filePath>.granges.rds) will overwritten.
keepStandardChr	TRUE/FALSE (default:TRUE). If it is set to TRUE, seqlevelsStyle will be converted to 'UCSC' and keepStandardChromosomes function will be applied to only keep data from the standard chromosomes.
...	Other arguments passed to rtracklayer::import.gff function

Value

A GRanges object containing the coordinates of the annotated genomic features in an input GTF file

Examples

```
#import the data and write it into a .rds file
## Not run:
importGtf(filePath='./Ensembl75.hg19.gtf')

## End(Not run)
#import the data but don't save it as RDS
## Not run:
importGtf(filePath='./Ensembl75.hg19.gtf', saveObjectAsRds = FALSE)

## End(Not run)
#import the data and overwrite the previously generated
## Not run:
importGtf(filePath='./Ensembl75.hg19.gtf', overwriteObjectAsRds = TRUE)

## End(Not run)
```

---

parseMsigdb *parseMsigdb*

Description

A function to import gene sets downloaded from the Molecular Signatures Database (MSIGDB)

Usage

```
parseMsigdb(filePath)
```

Arguments

filePath	Path to a file containing gene sets from MSIGDB. The gene ids must be in Entrez format.

Value

A list of vectors where each vector consists of a set of Entrez gene ids

Examples

```
#First Download gene sets (with Entrez Ids) from MSIGDB database
#from \url{http://software.broadinstitute.org/gsea/msigdb/collections.jsp#C2}

input <- system.file('extdata', 'msigdb_test.gmt', package='RCAS')
msigDB <- parseMsigdb (filePath = input)
```

---

plotFeatureBoundaryCoverage

plotFeatureBoundaryCoverage

Description

This function is used to create interactive plots displaying 5' and 3' end coverage profiles of given transcript features.

Usage

```
plotFeatureBoundaryCoverage(cvgF, cvgT, featureName)
```

Arguments

cvgF	data.frame object containing 'fiveprime' coverage data returned by getFeatureBoundaryCoverage function
cvgT	data.fram object containing 'threeprime' coverage data returned by getFeatureBoundaryCoverage function
featureName	character object. This is used to label the axes (e.g. transcripts, exons)

Value

a plotly htmlwidget is returned

Examples

```
data(queryRegions)
data(gff)
txdb <- GenomicFeatures::makeTxDbFromGRanges(gff)
transcriptCoords <- GenomicFeatures::transcripts(txdb)
cvgF <- getFeatureBoundaryCoverage (queryRegions = queryRegions,
                                    featureCoords = transcriptCoords,
                                    flankSize = 100,
                                    boundaryType = 'fiveprime',
```

```
                                    sampleN = 1000)

cvgT <- getFeatureBoundaryCoverage (queryRegions = queryRegions,
                                    featureCoords = transcriptCoords,
                                    flankSize = 100,
                                    boundaryType = 'threeprime',
                                    sampleN = 1000)
p <- plotFeatureBoundaryCoverage(cvgF = cvgF,
                                 cvgT = cvgT,
                            featureName = 'transcript')
```

`printMsigdbDataset` *Print MSIGDB Dataset to a file This function is used to print a MSIGDB dataset into a file. Mostly useful when human data is mapped to another species, and that mapping is required to run the report.*

Description

Print MSIGDB Dataset to a file This function is used to print a MSIGDB dataset into a file. Mostly useful when human data is mapped to another species, and that mapping is required to run the report.

Usage

```
printMsigdbDataset(dataset, outputFilename)
```

Arguments

`dataset`	A list of vectors containing gene sets from MSIGDB
`outputFilename`	A character string that denotes the output file name

Value

A text file printed to the current directory

Examples

```
data(geneSets)
printMsigdbDataset(geneSets, 'output.gmt')
```

queryGff *queryGff*

Description

This function checks overlaps between the regions in input query and in reference. Input query should be in BED format and reference should be in GFF format. Both data are imported as GRanges object.

Usage

```
queryGff(queryRegions, gffData)
```

Arguments

queryRegions	GRanges object imported from a BED file using importBed function
gffData	GRanges object imported from a GTF file using importGtf function

Value

a GRanges object (a subset of input gff) with an additional column 'overlappingQuery' that contains the coordinates of query regions that overlap the target annotation features

Examples

```
data(queryRegions)
data(gff)
overlaps <- queryGff(queryRegions = queryRegions, gffData = gff)
```

queryRegions *Sample BED file imported as a GRanges object*

Description

This dataset contains a randomly selected sample of human LIN28A protein binding sites detected by HITS-CLIP analysis downloaded from DoRina database (LIN28A HITS-CLIP hESCs (Wilbert 2012)). The BED file is imported via the importBed function and a subset of the data is selected by randomly choosing 10000 regions.

Usage

```
queryRegions
```

Format

GRanges object with 10000 ranges and 2 metadata columns

Value

A GRanges object

Source

http://dorina.mdc-berlin.de/regulators

`retrieveOrthologs` *retrieveOrthologs*

Description

Given two biomart connections and a set of entrez gene identifiers; retrieve orthologs between mart1 and mart2 for the given list of genes

Usage

```
retrieveOrthologs(mart1, mart2, geneSet)
```

Arguments

`mart1`	An Ensembl biomart connection for reference species created using the `biomaRt::useMart()` function
`mart2`	An Ensembl biomart connection for target species created using the `biomaRt::useMart()` function
`geneSet`	A vector of Entrez gene ids from a reference species (should be available at the biomart object, mart1)

Value

A data.frame object containing a mapping of orthologouse genes from two mart objects

Examples

```
mart1_hg19 <- biomaRt::useMart(biomart = 'ENSEMBL_MART_ENSEMBL',
                                  host = 'feb2014.archive.ensembl.org',
                               dataset = "hsapiens_gene_ensembl")
mart2_mm9 <- biomaRt::useMart(biomart = 'ENSEMBL_MART_ENSEMBL',
                                host = 'may2012.archive.ensembl.org',
                             dataset = "mmusculus_gene_ensembl")
genes <- c('2645','5232', '5230','5162','5160')
orthologs <- retrieveOrthologs( mart1 = mart1_hg19,
                                mart2 = mart2_mm9,
                                geneSet = genes)
```

runGSEA *runGSEA*

Description

This function is used to facilitate gene set enrichment analysis (GSEA) for a given set of genes

Usage

```
runGSEA(geneSetList, species = "human", backgroundGenes, targetedGenes)
```

Arguments

geneSetList	A named list of vectors where each vector consists of a set of Entrez gene ids (for instance, returned by parseMsigdb function)
species	A character string denoting the species under analysis. Options are 'human', 'mouse', 'fly' and 'worm'.
backgroundGenes	A vector of Ensembl gene ids that serve as background set of genes for GO term enrichment. In the context of RCAS, this should be the whole set of genes found in an input GTF file.
targetedGenes	A vector of Ensembl gene ids that serve as the set for which GSEA should be carried out. In the context of RCAS, this should be the set of genes that overlap the query regions

Value

A data.frame object containing enriched gene sets and associated statistics

Examples

```
#load test data
data(geneSets)
data(gff)
data(queryRegions)
#get all genes from the gff data
backgroundGenes <- unique(gff$gene_id)
#get genes that overlap query regions
overlaps <- queryGff(queryRegions, gff)
targetedGenes <- unique(overlaps$gene_id)
resultsGSEA <- runGSEA(geneSetList = geneSets,
          species = 'human',
          backgroundGenes = backgroundGenes,
          targetedGenes = targetedGenes)
```

runMotifRG *runMotifRG*

Description

This function makes use of motifRG library to carry out de novo motif discovery from input query regions

Usage

```
runMotifRG(queryRegions, resizeN = 0, sampleN = 0, genomeVersion,
  motifN = 5, nCores = 4)
```

Arguments

queryRegions	GRanges object containing coordinates of input query regions imported by the importBed function
resizeN	Integer value (default: 0) to resize query regions if they are shorter than the value of resize. Set to 0 to disable resize.
sampleN	A positive integer value. The queryRegions are randomly downsampled to include intervals as many as sampleN. The input will be downsampled only if this value is larger than zero and less than the total number of input intervals.
genomeVersion	A character string to denote the BS genome library required to extract sequences. Available options are hg19, mm9, ce10 and dm3.
motifN	A positive integer (default:5) denoting the maximum number of motifs that should be sought by the motifRG::findMotifFgBg function
nCores	A positive integer (default:4) number of cores used for parallel execution.

Value

a list of objects returned by the motifRG::findMotif function

Examples

```
data(queryRegions)
motifResults <- runMotifRG(queryRegions = queryRegions,
                           genomeVersion = 'hg19',
                           resize = 15,
                           motifN = 1,
                           nCores = 2)
```

runReport *Generate a RCAS Report for a list of transcriptome-level segments*

Description

This is the main report generation function for RCAS. This function can take a BED file, a GTF file and optionally an MSIGDB gene set annotation (or any text file containing annotations with the same structure as defined in MSIGDB); and use these input to run multiple RCAS functions to create a summary report regarding the annotation data that overlap the input BED file, enrichment analysis for GO terms, gene sets from MSIGDB, and motif analysis.

Usage

```
runReport(queryFilePath = "testdata", gffFilePath = "testdata",
  msigdbFilePath = "testdata", annotationSummary = TRUE,
  goAnalysis = TRUE, msigdbAnalysis = TRUE, motifAnalysis = TRUE,
  genomeVersion = "hg19", outDir = getwd(), printProcessedTables = FALSE,
  sampleN = 0, quiet = FALSE, selfContained = TRUE)
```

Arguments

queryFilePath	a BED format file which contains genomic coordinates of protein-RNA binding sites
gffFilePath	A GTF format file which contains genome annotations (preferably from ENSEMBL)
msigdbFilePath	Gene set annotations for Homo sapiens from Molecular Signatures Database or any text file that has the same structure. Regardless of which species is being studied (see genomeVersion parameter), msigdbFilePath must contain annotations for human genes. The gene sets will be mapped from human to other species if genomeVersion is set to anything except human genome versions (e.g. mm9 or dm3).
annotationSummary	TRUE/FALSE (default: TRUE) A switch to decide if RCAS should provide annotation summaries from overlap operations
goAnalysis	TRUE/FALSE (default: TRUE) A switch to decide if RCAS should run GO term enrichment analysis
msigdbAnalysis	TRUE/FALSE (default: TRUE) A switch to decide if RCAS should run gene set enrichment analysis
motifAnalysis	TRUE/FALSE (default: TRUE) A switch to decide if RCAS should run motif analysis
genomeVersion	A character string to denote for which genome version the analysis is being done. Available options are hg19/hg38 (human), mm9/mm10 (mouse), ce10 (worm) and dm3 (fly).
outDir	Path to the output directory. (default: current working directory)
printProcessedTables	boolean value (default: FALSE). If set to TRUE, raw data tables that are used for plots/tables will be printed to text files.

sampleN	integer value (default: 0). A parameter to determine if the input query regions should be downsampled to a smaller size in order to make report generation quicker. When set to 0, downsampling won't be done. To activate the sampling a positive integer value that is smaller than the total number of query regions should be given.
quiet	boolean value (default: FALSE). If set to TRUE, progress bars and chunk labels will be suppressed while knitting the Rmd file.
selfContained	boolean value (default: TRUE). By default, the generated html file will be self-contained, which means that all figures and tables will be embedded in a single html file with no external dependencies (See rmarkdown::html_document)

Value

An html generated using rmarkdown/knitr/pandoc that contains interactive figures, tables, and text that provide an overview of the experiment

Examples

```
#Default run will generate a report using built-in test data for hg19 genome.
## Not run:
runReport()

## End(Not run)

#A custom run for human
## Not run:
runReport( queryFilePath = 'input.BED',
           gffFilePath = 'annotation.gtf',
           msigdbFilePath = 'human_msigdb.gmt')

## End(Not run)
# To turn off certain modules of the report
## Not run:
runReport( queryFilePath = 'input.BED',
           gffFilePath = 'annotation.gtf',
           msigdbFilePath = 'human_msigdb.gmt',
           motifAnalysis = FALSE,
           goAnalysis = FALSE )

## End(Not run)
# To run the pipeline for species other than human
# If the msigdb module is needed, the msigdbFilePath
# must be set to the MSIGDB annotations for 'human'.
# MSIGDB datasets for other species will be calculated
# in the background using the createOrthologousMsigdbDataset
# function
## Not run:
runReport( queryFilePath = 'input.mm9.BED',
           gffFilePath = 'annotation.mm9.gtf',
           msigdbFilePath = 'msigdb.human.gmt',
           genomeVersion = 'mm9' )

## End(Not run)
```

runReportMetaAnalysis *runReportMetaAnalysis*

Description

Generate a stand-alone HTML report with interactive figures and tables from a pre-calculated RCAS database (using RCAS::createDB) to compare multiple samples.

Usage

```
runReportMetaAnalysis(dbPath = "RCAS.sqlite", sampleTablePath,
  outDir = getwd(), outFile = NULL, quiet = FALSE, selfContained = TRUE)
```

Arguments

dbPath	Path to the sqlite database generated by RCAS::createDB
sampleTablePath	A tab-separated file with two columns (no rownames) header 1: sampleName, header 2: sampleGroup
outDir	Path to the output directory. (default: current working directory)
outFile	Name of the output HTML report (by default, the base name of sampleTablePath value is used to create a name for the HTML report)
quiet	boolean value (default: FALSE). If set to TRUE, progress bars and chunk labels will be suppressed while knitting the Rmd file.
selfContained	boolean value (default: TRUE). By default, the generated html file will be self-contained, which means that all figures and tables will be embedded in a single html file with no external dependencies (See rmarkdown::html_document)

Value

An html generated using rmarkdown/knitr/pandoc that contains interactive figures, tables, and text that provide an overview of the experiment

Examples

```
dbPath <- system.file("extdata", "hg19.RCASDB.sqlite",
package='RCAS')

#Hint: use RCAS::summarizeDatabaseContent to see which samples have processed
#data in the database.
summarizeDatabaseContent(dbPath = dbPath)

#Create a data table for samples and their groups sampleGroup field is used
#to group replicates of #the same sample into one group in visualizations.
#Any arbitrary name can be used for sampleGroup field. However, entries in
#the sampleName field must be available in the queried database
sampleData <- data.frame('sampleName' = c('FUS', 'FMR1'),
    'sampleGroup' = c('FUS', 'FMR1'), stringsAsFactors = FALSE)
write.table(sampleData, 'sampleDataTable.tsv', sep = '\t',
    quote =FALSE, row.names = FALSE)
#Use the generated database to run a report
runReportMetaAnalysis(dbPath = 'hg19.RCASDB.sqlite',
    sampleTablePath = 'sampleDataTable.tsv')
```

runTopGO *runTopGO*

Description

A wrapper function to facilitate GO term enrichment analysis using topGO package

Usage

```
runTopGO(ontology = "BP", species = "human", backgroundGenes, targetedGenes)
```

Arguments

ontology	A character string denoting which type of GO ontology to use. Options are BP (biological processes), MF (molecular functions) and CC (cellular compartments).
species	A character string denoting which species is under analysis. Options are 'human', 'mouse', 'fly' and 'worm'.
backgroundGenes	A vector of Ensembl gene ids that serve as background set of genes for GO term enrichment. In the context of RCAS, this should be the whole set of genes found in an input GTF data.
targetedGenes	A vector of Ensembl gene ids that serve as the set for which GO term enrichment should be carried out. In the context of RCAS, this should be the set of genes that overlap with the query regions in an input BED file.

Value

A data.frame object containing enriched GO terms and associated statistics

Examples

```
## Not run:
#get all genes from the gff data
data(gff)
data(queryRegions)
backgroundGenes <- unique(gff$gene_id)
#get genes that overlap query regions
overlaps <- queryGff(queryRegions, gff)
targetedGenes <- unique(overlaps$gene_id)
#run TopGO
goResults = runTopGO( ontology = 'BP',
                      species = 'human',
                      backgroundGenes = backgroundGenes,
                      targetedGenes = targetedGenes)

## End(Not run)
```

summarizeDatabaseContent *summarizeDatabaseContent*

Description

Given a path to an sqlite database created using RCAS::createDB function, accesses the database and provides a quick summary of available samples and number of entries of each sample in the available tables of the database.

Usage

```
summarizeDatabaseContent(dbPath)
```

Arguments

dbPath	Path to the sqlite database

Value

A data.frame object

summarizeQueryRegions *summarizeQueryRegions*

Description

This function counts number of query regions that overlap with different types of gene features.

Usage

```
summarizeQueryRegions(queryRegions, txdbFeatures)
```

Arguments

queryRegions	GRanges object imported from a BED file using importBed function
txdbFeatures	List of GRanges objects - outputs of getTxdbFeaturesFromGRanges and getTxdbFeatures functions

Value

A data frame with two columns where first column holds features and second column holds corresponding counts

Examples

```
data(gff)
data(queryRegions)
txdbFeatures <- getTxdbFeaturesFromGRanges(gffData = gff)
summary <- summarizeQueryRegions(queryRegions = queryRegions,
                                 txdbFeatures = txdbFeatures)
```

summarizeQueryRegionsMulti
summarizeQueryRegionsMulti

Description

This function is a wrapper function to run RCAS::summarizeQueryRegions multiple times, which is useful to get a matrix of overlap counts between a list of BED files with a txdbFeatures extracted from GTF file

Usage

```
summarizeQueryRegionsMulti(queryRegionsList, txdbFeatures, nodeN = 1)
```

Arguments

queryRegionsList	GRangesList object imported from multiple BED files using `importBedFiles` function
txdbFeatures	List of GRanges objects - outputs of `getTxdbFeaturesFromGRanges` and `getTxdbFeatures` functions
nodeN	Positive integer value that denotes the number of cpus to use for parallel processing (default: 1)

Value

A list consisting of two data.frame objects: one for raw overlap counts and one for percentage of overlap counts (raw overlap counts divided by the number of query regions in the corresponding BED file)

Examples

```
data(gff)
data(queryRegions)
queryRegionsList <- GRangesList(queryRegions, queryRegions)
names(queryRegionsList) <- c('q1', 'q2')
txdbFeatures <- getTxdbFeaturesFromGRanges(gffData = gff)
summaryMatrix <- summarizeQueryRegionsMulti(queryRegionsList = queryRegionsList,
                                txdbFeatures = txdbFeatures)
```

Index