Package: RCAS (via r-universe)

November 25, 2026

Type Package

Title RNA Centric Annotation System

Version 1.40.0

Date 2024-03-19

Description RCAS is an R/Bioconductor package designed as a generic reporting tool for the functional analysis of transcriptome-wide regions of interest detected by high-throughput experiments. Such transcriptomic regions could be, for instance, signal peaks detected by CLIP-Seq analysis for protein-RNA interaction sites, RNA modification sites (alias the epitranscriptome), CAGE-tag locations, or any other collection of query regions at the level of the transcriptome. RCAS produces in-depth annotation summaries and coverage profiles based on the distribution of the query regions with respect to transcript features (exons, introns, 5'/3' UTR regions, exon-intron boundaries, promoter regions). Moreover, RCAS can carry out functional enrichment analyses and discriminative motif discovery.

License Artistic-2.0

LazyData TRUE

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

Imports GenomicRanges, IRanges, BSgenome, BSgenome.Hsapiens.UCSC.hg19, GenomeInfoDb (>= 1.12.0), Biostrings, rtracklayer, GenomicFeatures, txdbmaker, rmarkdown (>= 0.9.5), genomation (>= 1.5.5), knitr (>= 1.12.3), BiocGenerics, S4Vectors, plotrix, pbapply, RSQLite, proxy, pheatmap, ggplot2, cowplot, seqLogo, utils, ranger, gprofiler2

RoxygenNote 7.3.1

Suggests testthat, covr, BiocManager

SystemRequirements pandoc (>= 1.12.3)

VignetteBuilder knitr

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

Config/pak/sysreqs cmake make libbz2-dev libicu-dev liblzma-dev libpng-dev libuv1-dev libxml2-dev libssl-dev libx11-dev xz-utils zlib1g-dev

Repository Bioconductor 3.24

Date/Publication 2026-11-23 22:37:41 UTC

RemoteUrl https://github.com/bioc/RCAS

RemoteRef RELEASE_3_24

RemoteSha 19b28ec349738d827c20e6e22994042b05f2e814

Contents

calculateCoverageProfile . . . 3
calculateCoverageProfileFromTxdb . . . 4
calculateCoverageProfileList . . . 4
calculateCoverageProfileListFromTxdb . . . 5
checkSeqDb . . . 6
createControlRegions . . . 6
createDB . . . 7
createOrthologousGeneSetList . . . 8
deleteSampleDataFromDB . . . 9
discoverFeatureSpecificMotifs . . . 9
extractSequences . . . 10
findDifferentialMotifs . . . 11
findEnrichedFunctions . . . 12
generateKmers . . . 13
getFeatureBoundaryCoverage . . . 13
getFeatureBoundaryCoverageBin . . . 14
getFeatureBoundaryCoverageMulti . . . 15
getIntervalOverlapMatrix . . . 16
getMotifSummaryTable . . . 17
getPWM . . . 18
getTargetedGenesTable . . . 18
getTxdbFeatures . . . 19
getTxdbFeaturesFromGRanges . . . 19
gff . . . 20
importBed . . . 21
importBedFiles . . . 21
importGtf . . . 22
parseMsigdb . . . 23
plotFeatureBoundaryCoverage . . . 24
printMsigdbDataset . . . 25
queryGff . . . 25
queryRegions . . . 26
retrieveOrthologs . . . 26
runGSEA . . . 26
runMotifDiscovery . . . 27
runMotifRG . . . 28

runReport 28
runReportMetaAnalysis 30
runTopGO 31
summarizeDatabaseContent 32
summarizeQueryRegions 32
summarizeQueryRegionsMulti 33

Index **34**

`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 is deprecated. Use ?calculateCoverageProfile instead.

Usage

```
calculateCoverageProfileFromTxdb()
```

`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 is deprecated. Use ?calculateCoverageProfileList instead.

Usage

```
calculateCoverageProfileListFromTxdb()
```

checkSeqDb *checkSeqDb*

Description

Given a string that denotes a genome version (e.g. hg19) returns the BSgenome object matching the genome version that are available in BSgenome::available.genomes()

Usage

```
checkSeqDb(genomeVersion)
```

Arguments

genomeVersion	String that denotes genome version. To unambigously select a BSgenome object, provide a string that matches the end of the available genomes at: BSgenome::available.genomes().

Value

Returns a BSgenome object that uniquely matches the genomeVersion.

Examples

```
checkSeqDb('hg19')
```

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 This function is deprecated. For functional enrichment analysis, use findEnrichedFunctions.

Description

createOrthologousMsigdbDataset This function is deprecated. For functional enrichment analysis, use findEnrichedFunctions.

Usage

```
createOrthologousGeneSetList()
```

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')
```

`findDifferentialMotifs` *Find Differential Motifs*

Description

Find Differential Motifs

Usage

```
findDifferentialMotifs(
  querySeqs,
  controlSeqs,
  motifWidth = 6,
  motifN = 1,
  nCores = 1,
  maxMismatch = 1
)
```

Arguments

`querySeqs`	A DNAStringSet object that is the regions of interest.
`controlSeqs`	A DNAStrintSet object that serve as the control
`motifWidth`	A Positive integer (default: 6) for the generated k-mers. Warning: we recommend using values below 10 as the computation gets exponentially difficult as the motif width is increased.
`motifN`	A positive integer (default:1) denoting the maximum number of motifs that should be returned by the `findDifferentialMotifs` function
`nCores`	A positive integer (default:1) number of cores used for parallel execution.
`maxMismatch`	A positive integer (default: 1) - maximum number of mismatches to allow when searching for k-mer matches in sequences.

Examples

```
data(queryRegions)

# get query and control sequences
querySeqs <- extractSequences(queryRegions[1:500], 'hg19')
controlRegions <- createControlRegions(queryRegions[1:500])
controlSeqs <- extractSequences(controlRegions, 'hg19')

#run motif discovery
motifResults <- findDifferentialMotifs(querySeqs = querySeqs,
                                       controlSeqs = controlSeqs,
                                       motifWidth = 5,
                                       motifN = 1,
                                       maxMismatch = 0,
                                       nCores = 1)
#summarize motif results
getMotifSummaryTable(motifResults)
```

---

findEnrichedFunctions *findEnrichedFunctions*

Description

Find enriched functional terms among the genes that overlap the regions of interest.

Usage

```
findEnrichedFunctions(targetGenes, species, ...)
```

Arguments

targetGenes	Vector of Ensembl gene ids or gene names
species	First letter of genus + species name: e.g. hsapiens
...	Other arguments to be passed to gprofiler2::gost

Details

This function is basically a call to gprofiler2::gost function. It is here to serve as a replacement for other deprecated functional enrichment functions.

Examples

```
data(gff)
data(queryRegions)

overlaps <- queryGff(queryRegions, gff)
res <- findEnrichedFunctions(unique(overlaps$gene_id), 'hsapiens')
```

generateKmers *Generate K-mers*

Description

Given a list of characters, generates all possible fixed length strings

Usage

```
generateKmers(k, letters = c("A", "C", "G", "T"))
```

Arguments

k	The length of the strings to be generated
letters	A character vector

Value

Vector of strings

Examples

```
generateKmers(3, c('A', 'C', 'G'))
```

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 <- txdbmaker::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 <- txdbmaker::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 <- GenomicRanges::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

Get summary stats for top discovered motifs

Usage

```
getMotifSummaryTable(motifResults)
```

Arguments

`motifResults`	Output object of `runMotifDiscovery` function

Value

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

Examples

```
data(queryRegions)
motifResults <- runMotifDiscovery(queryRegions = queryRegions[1:1000],
                                  genomeVersion = 'hg19',
                                  resize = 15,
                                  motifN = 1,
                                  maxMismatch = 1,
                                  nCores = 2)
motifSummary <- getMotifSummaryTable(motifResults)
```

getPWM *getPWM*

Description

Given a vector of strings of equal width, generate a position-specific weight matrix based on the frequency of occurrence of the unique letters found in the sequences.

Usage

```
getPWM(sequences, letters = c("A", "C", "G", "T"))
```

Arguments

sequences	vector of strings of equal widths.
letters	vector of characters to consider as the an alphabet

Value

A matrix of position-specific-weights.

Examples

```
sequences = c("GGAGAG", "GAAGAA", "TGAGAA", "GGAGAA", "GAAGAA")
getPWM(sequences)
```

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 <- txdbmaker::makeTxDbFromGRanges(gff)
txdbFeatures <- getTxdbFeatures(txdb)
featuresTable <- getTargetedGenesTable(queryRegions = queryRegions,
                                       txdbFeatures = txdbFeatures)

## End(Not run)
```

---

getTxdbFeatures *getTxdbFeatures*

Description

This function is deprecated. Use getTxdbFeaturesFromGRanges instead.

Usage

```
getTxdbFeatures()
```

---

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

This function is deprecated. For functional enrichment analysis, use findEnrichedFunctions.

Usage

```
parseMsigdb()
```

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 <- txdbmaker::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*

Description

This function is deprecated. For functional enrichment analysis, use findEnrichedFunctions.

Usage

```
printMsigdbDataset()
```

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

This function is deprecated. For functional enrichment analysis, use findEnrichedFunctions.

Usage

```
retrieveOrthologs()
```

runGSEA *runGSEA*

Description

This function is deprecated. For functional enrichment analysis, use findEnrichedFunctions.

Usage

```
runGSEA()
```

runMotifDiscovery *runMotifDiscovery*

Description

This function builds a random forest classifier to find the top most discriminative motifs in the query regions compared to the background. The background sequences are automatically generated based on the query regions. First, k-mers of a fixed length are generated. The query and control sequences are searched for k-mers allowing for mismatches. A random forest model is trained to find the most discriminative motifs.

Usage

```
runMotifDiscovery(
  queryRegions,
  resizeN = 0,
  motifWidth = 6,
  sampleN = 0,
  genomeVersion,
  maxMismatch = 1,
  motifN = 5,
  nCores = 1
)
```

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.
motifWidth	A Positive integer (default: 6) for the generated k-mers. Warning: we recommend using values below 10 as the computation gets exponentially difficult as the motif width is increased.
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. Example: 'hg19'
maxMismatch	A positive integer (default: 1) - maximum number of mismatches to allow when searching for k-mer matches in sequences.
motifN	A positive integer (default:5) denoting the maximum number of motifs that should be returned by the findDifferentialMotifs function
nCores	A positive integer (default:1) number of cores used for parallel execution.

Value

A list of four objects: k-mer count matrices for query and background and lists of string matches for the top discriminating motifs (motifN).

Examples

```
data(queryRegions)
motifResults <- runMotifDiscovery(queryRegions = queryRegions[1:1000],
                                  genomeVersion = 'hg19',
                                  motifWidth = 6,
                                  resize = 15,
                                  motifN = 1,
                                  maxMismatch = 1,
                                  nCores = 1)
```

`runMotifRG` *run motifRG*

Description

run motifRG

Usage

```
runMotifRG()
```

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

Description

This is the main report generation function for RCAS. This function takes a BED file, a GTF file to create a summary report regarding the annotation data that overlap the input BED file, enrichment analysis for functional terms, and motif analysis.

Usage

```
runReport(
  queryFilePath = "testdata",
  gffFilePath = "testdata",
  annotationSummary = TRUE,
  goAnalysis = 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)
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
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.
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',
           genomeVersion = 'hg19')

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

## 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

This function is deprecated. Use findEnrichedFunctions instead.

Usage

```
runTopGO()
```

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

Index

∗ **datasets**
gff, 20
queryRegions, 26

calculateCoverageProfile, 3
calculateCoverageProfileFromTxdb, 4
calculateCoverageProfileList, 4
calculateCoverageProfileListFromTxdb, 5
checkSeqDb, 6
createControlRegions, 6
createDB, 7
createOrthologousGeneSetList, 8

deleteSampleDataFromDB, 9
discoverFeatureSpecificMotifs, 9

extractSequences, 10

findDifferentialMotifs, 11
findEnrichedFunctions, 12

generateKmers, 13
getFeatureBoundaryCoverage, 13
getFeatureBoundaryCoverageBin, 14
getFeatureBoundaryCoverageMulti, 15
getIntervalOverlapMatrix, 16
getMotifSummaryTable, 17
getPWM, 18
getTargetedGenesTable, 18
getTxdbFeatures, *10*, *18*, 19
getTxdbFeaturesFromGRanges, *10*, *18*, 19
gff, 20

importBed, *6*, *10*, *18*, 21, *27*
importBedFiles, 21
importGtf, 22

parseMsigdb, 23
plotFeatureBoundaryCoverage, 24
printMsigdbDataset, 25
queryGff, 25
queryRegions, 26

retrieveOrthologs, 26
runGSEA, 26
runMotifDiscovery, 27
runMotifRG, *10*, 28
runReport, 28
runReportMetaAnalysis, 30
runTopGO, 31

summarizeDatabaseContent, 32
summarizeQueryRegions, 32
summarizeQueryRegionsMulti, 33