Package 'kmeRtone'

March 8, 2024

Type Package

Version 1.0

Date 2024-02-09

Title kmeRtone

Description Multi-purpose and transferrable k-meric enrichment/depletion analysis software.

SystemRequirements GNU make

Imports data.table (>= 1.15.0),
R6 (>= 2.5.1),
Rcpp (>= 1.0.12),
R.utils (>= 2.12.3),
openxlsx (>= 4.2.5.2),
png (>= 0.1-8),
RcppSimdJson (>= 0.1.11),
venneuler (>= 1.1-4),
stringi,
curl,
future,
future.apply,
jsonlite,
progressr,
Biostrings,
seqLogo

Depends R (>= 4.2)

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

LinkingTo Rcpp, stringi, Rhtslib, zlibbioc

URL https://github.com/SahakyanLab/kmeRtone

BugReports https://github.com/SahakyanLab/kmeRtone/issues

Encoding UTF-8

License GPL-3 + file LICENSE

LazyData true

Suggests BiocStyle,
knitr,
rmarkdown,
testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

R topics documented:

addAlphaCol . . . 4
bedToCoor . . . 4
buildControl . . . 5
buildKmerTable . . . 5
buildMidPatternKmerTable . . . 6
buildRangedKmerTable . . . 6
buildRESTurl . . . 7
calKmerSkew . . . 8
calPWM . . . 8
catHeader . . . 9
Coordinate . . . 9
countBaseComposition . . . 11
countChoppedKmers . . . 12
countDistribution . . . 13
countKmers . . . 13
countMidPatternContext . . . 14
countMidPatternContext2 . . . 14
countMidPatternKmers . . . 15
countMidPatternRangedKmers . . . 16
countPointContext . . . 16
countPointContext2 . . . 17
countRangedKmers . . . 17
countRevCompKmers . . . 18
countSlidingWindow . . . 18
countSlidingWindow2 . . . 19
count_substring_fixed . . . 19
count_substring_regex . . . 20
downloadNCBIGenomes . . . 20
downloadUCSCgenome . . . 21
example_genome_coor . . . 21
example_kmeRtone_score . . . 22
EXPLORE . . . 23
extractKmers . . . 24
generateGenicElementCoor . . . 25
generateIntergenicCoor . . . 26
getCOSMICauthURL . . . 27
getCOSMICcancerGeneCensus . . . 27
getCOSMIClatestVersion . . . 28
getCOSMICmutantExport . . . 28
getEnsemblData . . . 29
getEnsemblRegionFeatures . . . 29
getEnsemblVariantFeatures . . . 30
getEnsemblVariantFeatures_serial . . . 31
getGnomADvariants . . . 31
getICTVvirusMetadataResource . . . 32
getNCBIassemblySummary . . . 33

getNCBItaxonomy . . . 33
getScores . . . 34
getUCSCgenePredTable . . . 34
getVCFmetainfo . . . 35
GET_OPTIMUM_K . . . 35
initKmerTable . . . 36
kmeRtone . . . 36
Kmer_Table . . . 39
loadCoordinate . . . 40
loadGenome . . . 41
loadGenomicContents . . . 42
mapKmers . . . 43
mergeCoordinate . . . 44
mixColors . . . 44
NCBI_Genome . . . 45
partitionCoordinate . . . 46
persistentDownload . . . 47
readBCF . . . 48
readBED . . . 48
readFASTA . . . 49
readSingleFASTA . . . 49
readVCF . . . 50
readVCF2 . . . 50
removeRegion . . . 51
reverseComplement . . . 51
SCORE . . . 52
scoreKmers . . . 53
selectGenomesForCrossSpeciesStudy . . . 54
selectRepresentativeFromASM . . . 55
simulatePopulation . . . 55
splitFASTA . . . 56
splitFASTA2 . . . 57
STUDY_ACROSS_POPULATIONS . . . 57
STUDY_ACROSS_SPECIES . . . 58
STUDY_CANCER_GENES . . . 59
STUDY_GENIC_ELEMENTS . . . 60
system3 . . . 61
trimCoordinate . . . 62
UCSC_Genome . . . 62
writeBED . . . 64
writeFASTA . . . 64
writeVCF . . . 65

Index **66**

addAlphaCol *Add transparency to color.*

Description

Add transparency to color.

Usage

```
addAlphaCol(cols, alpha)
```

Arguments

cols	Colors in hex format or R color code e.g. "red", "black", etc.
alpha	Alpha value.

Value

Colors with alpha value in hex format.

bedToCoor *Convert a BED file to chromosome-separated csv files.*

Description

Convert a BED file to chromosome-separated csv files.

Usage

```
bedToCoor(bed.path, output.path = "coordinate/", compress = TRUE)
```

Arguments

bed.path	A path to a BED file.
output.path	Output directory path. It should be an empty directory. Default to coordinate/
compress	Logical. If TRUE, compress the output CSV files. Default to TRUE.

Value

None

`buildControl` *Build control regions*

Description

Build control regions

Usage

```
buildControl(
  case,
  ctrl.rel.pos,
  genome,
  output.path = "control/",
  verbose = TRUE
)
```

Arguments

`case`	Case in Coordinate class object format.
`ctrl.rel.pos`	Control relative position.
`genome`	Genome class object.
`output.path`	Output directory path to save control coordinate.
`verbose`	Boolean. Default is TRUE and will print progress updates.

Value

Control in Coordinate class object format.

`buildKmerTable` *Count k-mers from given sequence(s) and build a data.table of k-mer counts.*

Description

Only existed k-mers are returned in data.table object.

Usage

```
buildKmerTable(dna.seqs, k, method = "auto", remove.N = TRUE)
```

Arguments

`dna.seqs`	String of sequence(s).
`k`	Size of kmer.
`method`	K-mer counting method: "Biostrings", "sliding", or "auto". Default is "auto"; For k > 8, sliding method is used.
`remove.N`	Remove unknown base? Default is TRUE.

Value

A `data.table` object with column kmer and N.

`buildMidPatternKmerTable` *Count k-mers with specified middle pattern from given sequence(s) and build a data.table of k-mer counts.*

Description

Only existed k-mers are returned in data.table object.

Usage

```
buildMidPatternKmerTable(dna.seqs, k, mid.patterns, remove.N = TRUE)
```

Arguments

`dna.seqs`	String of sequence(s).
`k`	Size of kmer.
`mid.patterns`	Middle patterns.
`remove.N`	Remove unknown base? Default is TRUE.

Value

A `data.table` object with column kmer and N.

`buildRangedKmerTable` *Count kmers from a sequence in given ranges and build a data.table of k-mer counts.*

Description

Count kmers from a sequence in given ranges and build a data.table of k-mer counts.

Usage

```
buildRangedKmerTable(
  dna.seq,
  starts,
  ends,
  k,
  method = "sliding",
  chopping.method = "auto",
  remove.N = TRUE
)
```

Arguments

dna.seq	String of sequence.
starts	Start positions.
ends	End positions.
k	Size of kmer.
method	Method options: "sliding" or "chopping". Chopping consumes a lot of memory for extremely long sequence using "substring" method. Using "Biostrings" for k > 12 is memory consuming. Default is "sliding".
chopping.method	Chopping method: "Biostrings" or "substring". Default is "auto".
remove.N	Remove unknown base N? Default is TRUE.

Value

A data.table object with column kmer and N.

---

buildRESTurl *Function constructs a URL for a REST API call by appending query parameters.*

Description

Function constructs a URL for a REST API call by appending query parameters.

Usage

```
buildRESTurl(url, .list = list(), ...)
```

Arguments

url	Base URL of the REST API.
.list	A list of named query parameters.
...	additional optional arguments

Value

string of the full REST API URL.

`calKmerSkew` *Function calculates the skew of k-mers based on their occurrence in positive and negative strands.*

Description

Function calculates the skew of k-mers based on their occurrence in positive and negative strands.

Usage

```
calKmerSkew(kmer.table)
```

Arguments

`kmer.table`	data.table with columns: kmer, pos_strand, neg_strand.

Value

data.table with the kmer_skew column.

`calPWM` *Calculate position weight matrix of overlapping sequences. Simulation of human population is based on single nucleotide variation.*

Description

Calculate position weight matrix of overlapping sequences. Simulation of human population is based on single nucleotide variation.

Usage

```
calPWM(
  kmers,
  pseudo.num = 0,
  bg.prop = c(a = 0.295, c = 0.205, g = 0.205, t = 0.295),
  output = "PWM"
)
```

Arguments

`kmers`	A vector of k-mers to overlap.
`pseudo.num`	Pseudo-number to avoid numerical instability due to lack of base at a position. Default is zero i.e. no pseudo-number.
`bg.prop`	Background proportion of bases. Default is c(a = 0.295, c = 0.205, g = 0.205, t = 0.295) which is observed in human genome.
`output`	Output matrix type. Options are PCM, PPM, and PWM which refer to position count/probability/weight matrix. Default is PWM.

Value

A position count/probability/weight matrix.

`catHeader` *Function prints a given message in a formatted header with borders.*

Description

Function prints a given message in a formatted header with borders.

Usage

```
catHeader(msg)
```

Arguments

`msg`	message to be printed within the header.

`Coordinate` *Function constructs an R6 object for handling coordinate data. The object includes methods for loading, manipulating, and analyzing coordinate data.*

Description

Function constructs an R6 object for handling coordinate data. The object includes methods for loading, manipulating, and analyzing coordinate data.

Function constructs an R6 object for handling coordinate data. The object includes methods for loading, manipulating, and analyzing coordinate data.

Public fields

`root_path` A path to a directory containing coordinate files.

`single_len` Single case length e.g. damage length. Default is NULL.

`is_strand_sensitive` Coordinate strand polarity. Default is TRUE.

`merge_replicate` Merge coordinate from different replicates. Default is TRUE.

`rm_dup` Remove duplicate entry in the coordinate table. Default is TRUE.

`add_col_rep` If add_col_rep is TRUE, column replicate is added to the coordinate table. Default is TRUE.

`paths` Individual coordinate files.

`rep_names` Replicate names determined from coordinate subdirectory.

`chr_names` Chromosome names determined from filenames.

`coor` Chromosome-named list of coordinate data.table.

`is_kmer` A data.table of is_kmer status. The first column is original is_kmer status.

`k` K-mer size when is_kmer is TRUE. When is_kmer is FALSE, k is NA.

`ori_first_index` Original chromosome-separated table first index is either starting from zero or one.

`load_limit` Maximum coordinate table loaded.

Methods

Public methods:

- Coordinate$new()
- Coordinate$[()
- Coordinate$mark_overlap()
- Coordinate$print()
- Coordinate$map_sequence()
- Coordinate$clone()

Method `new()`**:** Create a new Coordinate class

Usage:

```
Coordinate$new(
  root.path,
  single.len,
  is.strand.sensitive,
  merge.replicate,
  rm.dup,
  add.col.rep,
  is.kmer,
  k,
  ori.first.index,
  load.limit
)
```

Arguments:

`root.path` A path to a directory containing either: (1) chromosome-separated coordinate files (assume replicates for subdirectories) OR (2) bedfile. (assume replicates for bedfiles)

`single.len` Single case length e.g. damage length. Default is NULL

`is.strand.sensitive` A boolean whether strand polarity matters. Default is TRUE.

`merge.replicate` Merge coordinate from different replicates. Default is TRUE. If not merging, duplicates will give weight to the kmer counting. If add_col_rep, merged coordinate will contain column replicate e.g. "rep1&rep2".

`rm.dup` Remove duplicates in each replicate. Default is FALSE Default is FALSE

`add.col.rep` Add column replicate to coordinate table.

`is.kmer` Is the coordinate refers to k-mer i.e. expanded case? Default is FALSE.

`k` Length of k-mer if is_kmer is TRUE.

`ori.first.index` Zero- or one-based index. Default is 1.

`load.limit` Maximum coordinate data.table loaded. Default is 1.

Returns: A new `Coordinate` object.

Method `[()`**:** Calling coordinate table by loading on demand. Maximum load is determine by load_limit field.

Usage:

```
Coordinate$[(
  chr.name,
  state = "current",
  k,
  reload = FALSE,
  rm.other.cols = TRUE
)
```

Arguments:

`chr.name` Chromosome name. It can be a vector of chromosomes.

`state` Coordinate state: "current", "case", "kmer". The coordinate state is changed automatically on demand. Default is "current".

`k` K-mer size. If state is "kmer", k is needed to expand the coordinate.

`reload` Reload the coordinate table from the root.path. Default is TRUE.

`rm.other.cols` Remove unnecessary columns for kmeRtone operation.

Returns: A single or list of data.table coordinate of requested chromosome.

Method `mark_overlap()`**:** Mark overlapping regions in the coordinate table. A column name is_overlap is added.

Usage:

```
Coordinate$mark_overlap()
```

Arguments:

`chr.names` Chromosome names

Returns: New column is_overlap is added.

Method `print()`**:** Print `Coordinate` object parameters.

Usage:

```
Coordinate$print()
```

Returns: Message of `Coordinate` object parameters.

Method `map_sequence()`**:** Get corresponding sequence from the loaded coordinate.

Usage:

```
Coordinate$map_sequence(genome)
```

Arguments:

`genome` Genome object or vector of named chromosome sequences.

Returns: New column seq.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Coordinate$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`countBaseComposition` *Function performs an analysis of base composition including sequence frequency, PWM calculations, and G/C content at various window sizes.*

Description

Function performs an analysis of base composition including sequence frequency, PWM calculations, and G/C content at various window sizes.

Usage

```
countBaseComposition(case, genome, case.pattern, output.path = "./")
```

Arguments

case	A Coordinate class object or similar structure.
genome	Genome class object or similar structure.
case.pattern	String patterns to consider in the analysis.
output.path	Output path for saving the analysis results.

---

countChoppedKmers *Function chops k-mers within specified ranges of a sequence and counts them. It uses either a substring method or functionalities from the Biostrings package.*

Description

Function chops k-mers within specified ranges of a sequence and counts them. It uses either a substring method or functionalities from the Biostrings package.

Usage

```
countChoppedKmers(dna.seq, starts, ends, k, method = "auto")
```

Arguments

dna.seq	A string of sequence.
starts	Start positions.
ends	End positions.
k	Size of kmer.
method	Method: "Biostrings" or "substring". Default is Biostrings.

Value

A k-mer-named vector of counts.

countDistribution *Function performs an analysis of the distribution of genomic cases, including: A) Check case distribution in:*

1. *replicates*
2. *chromosomes*
3. *strands B) Check case base composition and filter out other case.patterns*
4. *draw seqLogo in 100 or 101 base context*
5. *calculate G+C percentage Then, it generates various plots like bar plots and Venn/Euler diagrams.*

Description

Function performs an analysis of the distribution of genomic cases, including: A) Check case distribution in:

1. replicates
2. chromosomes
3. strands B) Check case base composition and filter out other case.patterns
4. draw seqLogo in 100 or 101 base context
5. calculate G+C percentage Then, it generates various plots like bar plots and Venn/Euler diagrams.

Usage

```
countDistribution(case, genome, case.pattern, output.path = "./")
```

Arguments

case	A Coordinate class object or similar structure for genomic data.
genome	Genome class object or similar structure.
case.pattern	String patterns to consider in the analysis.
output.path	Output path for saving the analysis results.

countKmers *Count k-mers from string(s) using a simple hash table.*

Description

Count only observed k-mers. Biostrings::oligonucleotideFrequency reports all possible k-mers. For k > 12, the memory for creating empty k-mer counts spiked and crashed the R session.

Usage

```
countKmers(sequences, k)
```

Arguments

sequences	Sequence strings.
k	Size of k-mer.

Value

A vector of k-mer counts. The counts of multiple sequences are combined, similar to Biostrings::oligonucleotideFrequenc simplify.as "collapsed".

---

countMidPatternContext

Locate a middle sequence pattern and count its sequence context.

Description

Locate a middle sequence pattern and count its sequence context.

Usage

```
countMidPatternContext(sequence, mid_pattern, window, context_pattern)
```

Arguments

sequence	A sequence to slide.
mid_pattern	A middle pattern to search for.
window	Size of a surrounding window.
context_pattern	A context pattern to search for.

Value

A numeric vector of count.

---

countMidPatternContext2

Locate a middle sequence pattern and count its sequence context.

Description

This function searches for a specified middle pattern within a given sequence. It then counts the occurrences of specific context patterns within a defined window size around the middle pattern. The function returns a map where keys are the counts of context patterns found and values are the frequencies of these counts.

Usage

```
countMidPatternContext2(sequence, mid_pattern, window, context_patterns)
```

Arguments

sequence	A string representing the sequence to be analyzed.
mid_pattern	A string representing the middle pattern to search for within the sequence.
window	An integer specifying the size of the surrounding window around the middle pattern.
context_patterns	A vector of strings representing the context patterns to search for within the window.

Value

A std::unordered_map<int,int> where keys are the counts of context patterns found and values are the frequencies of these counts.

Examples

```
sequence <- "ATCGATCGA"
mid_pattern <- "CG"
window <- 5
context_patterns <- c("AT", "GA")
countMidPatternContext2(sequence, mid_pattern, window, context_patterns)
```

---

countMidPatternKmers *Count Relevant K-mers with Specified Middle Pattern from Sequence String(s)*

Description

This function scans through each sequence in the provided vector, locating a specified middle pattern. For each occurrence of the middle pattern, the function extracts and counts the surrounding k-mers. The k-mers are identified based on the given k-mer size and centered around the middle pattern.

Usage

```
countMidPatternKmers(sequences, k, mid_pattern)
```

Arguments

sequences	A vector of strings, each representing a sequence to be analyzed.
k	An integer specifying the size of the k-mers to be extracted and counted.
mid_pattern	A string representing the middle pattern to search for within each sequence.

Value

A std::unordered_map with k-mers as keys and their counts as values.

Examples

```
sequences <- c("ATCGATCGA", "GCGCATGCA")
k <- 5
mid_pattern <- "CG"
countMidPatternKmers(sequences, k, mid_pattern)
```

`countMidPatternRangedKmers` *Count k-mers in given ranges of a sequence.*

Description

Slide and update the cummulated table count.

Usage

```
countMidPatternRangedKmers(sequence, starts, ends, k, mid_pattern)
```

Arguments

`sequence`	A sequence to count.
`starts`	Start positions.
`ends`	End positions.
`k`	K-mer size.
`mid_pattern`	A middle pattern to search for.

Value

A k-mer-named vector of count.

`countPointContext` *Ccount sequence context of given point positions.*

Description

Ccount sequence context of given point positions.

Usage

```
countPointContext(sequence, points, len, window, context_pattern)
```

Arguments

`sequence`	A sequence to slide.
`points`	Middle point positions.
`len`	Length of the middle point. Default to 1.
`window`	Size of a surrounding window.
`context_pattern`	A context pattern to search for.

Value

A numeric vector of count.

`countPointContext2` *Ccount sequence context of given point positions.*

Description

Ccount sequence context of given point positions.

Usage

```
countPointContext2(sequence, points, len, window, context_patterns)
```

Arguments

`sequence`	A sequence to slide.
`points`	Middle point positions.
`len`	Length of the middle point.
`window`	Size of a surrounding window.
`context_patterns`	Context patterns to search for.

Value

A named vector of frequency of counts.

`countRangedKmers` *Count k-mers in given ranges of a sequence.*

Description

Slide and update the cummulated table count.

Usage

```
countRangedKmers(sequence, starts, ends, k)
```

Arguments

`sequence`	A sequence to count.
`starts`	Start positions.
`ends`	End positions.
`k`	K-mer size.

Value

A k-mer-named vector of count.

countRevCompKmers *Count reverse complement sequence from its opposite strand. Build for k-mer table generated from initKmerTable function but applicable to others with the same format.*

Description

Count reverse complement sequence from its opposite strand. Build for k-mer table generated from initKmerTable function but applicable to others with the same format.

Usage

```
countRevCompKmers(kmer.table)
```

Arguments

kmer.table	A data.table of k-mer with at least 3 columns: kmer, pos_strand, and neg_strand. Splitted k-mer columns: kmer_part1 and kmer_part2 is supported.

Value

Updated k-mer table.

countSlidingWindow *Count sequence content in a sliding window of a sequence.*

Description

Count sequence content in a sliding window of a sequence.

Usage

```
countSlidingWindow(sequence, window, pattern)
```

Arguments

sequence	A sequence to slide.
window	Size of a window.
pattern	A pattern to search for.

Value

A numeric vector of count.

countSlidingWindow2 *Count sequence content in a sliding window of a sequence.*

Description

Count sequence content in a sliding window of a sequence.

Usage

```
countSlidingWindow2(sequence, window, patterns)
```

Arguments

sequence	A sequence to slide.
window	Size of a window.
patterns	Patterns of the same size to search for.

Value

Named vector of frequency of count.

count_substring_fixed *Count sequence content in a given sequence.*

Description

stringi has no function that search within substring without memory copy it. This function has two versions. One without the need to memory copy denoted as ***. The only downside to this is std::string::find cannot stop searching past end of substring. I manage to at least stop it as soon as possible. If the pattern is long and rare, it won't stop until it find post-substring pattern. The other version is memory copy substring but as this operation is in the loop, the memory is still within comfortable range. c++17 has std::string_view that solve this but still new and not widely available. Use count_substring_regex to avoid memory copy.

Usage

```
count_substring_fixed(sequence, start, end, pattern)
```

Arguments

sequence	A sequence to map.
start	Start positions.
end	End positions.
pattern	A pattern to search for.

Value

A numeric vector of count.

`count_substring_regex` *Count sequence content in a given sequence.*

Description

stringi has no function that search within substring without memory creating it. This function solve that. Unlike count_substring_fixed, this function does not need to memory copy substring.

Usage

```
count_substring_regex(sequence, start, end, pattern)
```

Arguments

`sequence`	A sequence to map.
`start`	Start positions.
`end`	End positions.
`pattern`	A regex pattern to search for within start and end positions.

Value

A numeric vector of count.

`downloadNCBIGenomes` *Function downloads genome fasta files from the NCBI FTP database. Users can provide either organism names or an assembly summary data table.*

Description

Supports options for splitting multi-header fasta files and overwriting existing files.

Usage

```
downloadNCBIGenomes(
  asm,
  species,
  db,
  output.dir = "./",
  split.fasta = FALSE,
  overwrite = FALSE
)
```

Arguments

`asm`	NCBI assembly summary data.table
`species`	Species names.
`db`	Database record to use: refseq or genbank
`output.dir`	Output directory path. Default is current directory.
`split.fasta`	NCBI fasta files are multi-header. Split them? Default is FALSE.
`overwrite`	Overwrite any existed genome file? Default is FALSE to skip the download.

Value

Genome fasta file(s) named according to the FTP database convention.

`downloadUCSCgenome` *Function downloads chromosome-separated fasta genome sequences from the UCSC database. Users can specify a genome name, an output folder, and a specific chromosome or chromosomes. There's an option to choose the download method as well.*

Description

Function downloads chromosome-separated fasta genome sequences from the UCSC database. Users can specify a genome name, an output folder, and a specific chromosome or chromosomes. There's an option to choose the download method as well.

Usage

```
downloadUCSCgenome(genome.name, output.path, chr.name, method = "curl")
```

Arguments

`genome.name`	Genome name (e.g., hg19, hg38, mm19).
`output.path`	Output folder for the downloaded sequences.
`chr.name`	Specific chromosome to download; defaults to all if unspecified.
`method`	Download method for the `download.file` function.

Value

An output folder containing chromosome-separated fasta files.

`example_genome_coor` *Example genome coordinate file*

Description

Below is an example code that generates random genomic coordinates.

library(data.table) library(kmeRtone)

1. Randomly generate genomic positions and save results dir.create("./data", showWarnings = FALSE)

set.seed(1234) for(chr in 1) genomic_coor <- data.table::data.table(seqnames = paste0("chr", chr), start = sample(x = 10000:10000000, size = 100000, replace = FALSE), width = 2)

```
data.table::fwrite(
    genomic_coor,
    paste0("./data/chr", chr, ".csv")
)
```

Usage

```
example_genome_coor
```

Format

A data frame with 1001 rows and 3 columns

seqnames Chromosome number of the recorded biological event, e.g. DNA strand breaks

start 5' start position of the recorded biological event

width Sequence width of the recorded biological event, e.g. 2 for a DNA strand break

---

`example_kmeRtone_score`

Example 2-mer enrichment/depletion scores

Description

Below is an example code that generates random genomic coordinates and runs the default kmeRtone `SCORE` function to quantify the k-meric enrichment and depletion.

library(data.table) library(kmeRtone)

1. Randomly generate genomic positions and save results dir.create("./data", showWarnings = FALSE)

set.seed(1234) for(chr in 1:22) genomic_coor <- data.table::data.table(seqnames = paste0("chr", chr), start = sample(x = 10000:10000000, size = 100000, replace = FALSE), width = 2)

```
data.table::fwrite(
    genomic_coor,
    paste0("./data/chr", chr, ".csv")
)
```

#' 2. Run kmeRtone `score` function kmeRtone::kmeRtone(case.coor.path="./data", genome.name="hg19", strand.sensitive=FALSE, k=2, ctrl.rel.pos=c(80, 500), case.pattern=NULL, single.case.len=2, output.dir="output", module="score", rm.case.kmer.overlaps=FALSE, merge.replicate=TRUE, kmer.table=NULL, verbose=TRUE)

Usage

```
example_kmeRtone_score
```

Format

A data frame with 1001 rows and 3 columns

case Case k-mers, e.g. damage k-mer counts

case_skew Case k-mers skews, e.g. skew of the damage k-mers counts

control control k-mers, e.g. damage k-mer counts

control_skew control k-mers skews, e.g. skew of the damage k-mers counts

kmer K-meric sequence

z Intrinsic susceptibility z-score for each k-mer

Source

https://github.com/SahakyanLab/kmeRtone/tree/master/README.md

---

EXPLORE *Function generates various exploratory analyses.*

Description

Function generates various exploratory analyses.

Usage

```
EXPLORE(
  case.coor.path,
  genome.name,
  strand.sensitive,
  k,
  case.pattern,
  output.path,
  case,
  genome,
  control,
  genome.path,
  single.case.len,
  rm.dup,
  case.coor.1st.idx,
  coor.load.limit,
  genome.load.limit,
  genome.fasta.style,
  genome.ncbi.db,
  use.UCSC.chr.name,
  verbose
)
```

Arguments

case.coor.path	Path to case coordinates.
genome.name	Genome name (e.g., hg19, hg38).
strand.sensitive	Boolean indicating if strand sensitivity is considered.
k	K-mer size.
case.pattern	String patterns to consider in the analysis.
output.path	Output directory path for exploration plots.
case	Coordinate class object or similar structure for case data.
genome	Genome class object or similar structure.
control	Control class object or similar structure.
genome.path	Path to genome fasta files.

`single.case.len`
: Length of single cases.

`rm.dup`
: Boolean indicating if duplicates should be removed.

`case.coor.1st.idx`
: Indexing of case coordinates.

`coor.load.limit`
: Maximum number of coordinates to load.

`genome.load.limit`
: Maximum number of genome data to load.

`genome.fasta.style`
: Fasta file style for genome data.

`genome.ncbi.db`
: NCBI database for genome data.

`use.UCSC.chr.name`
: Boolean indicating if UCSC chromosome naming is used.

`verbose`
: Boolean indicating if verbose output is enabled.

Value

Output directory containing exploration plots.

`extractKmers` *Extract k-mers from a given Coordinate object and Genome objects*

Description

A k-mer table is initialized and updated in every chromosome-loop operation. There are 3 modes of extraction. (1) When k is smaller than 9 or k is larger than 15, the k-mer is extracted in a standard way. A k-mer table with every possible k-mers is created and updated. (2) For k between 9 and 13, the k-mer sequence is split to half to reduce memory usage significantly. e.g. ACGTACGTA will become ACGT ACGTA. (3) When k is larger than 14, k-mers are extracted the same way as (1) but the k-mer table is grown or expanded for every new k-mer found.

Usage

```
extractKmers(
  coor,
  genome,
  k,
  central.pattern = NULL,
  rm.overlap.region = TRUE,
  verbose = TRUE
)
```

Arguments

`coor`
: Coordinate class object.

`genome`
: Genome class object.

`k`
: Length of k-mer.

`central.pattern`
: Central pattern of the k-mer, if applicable.

`rm.overlap.region`
: Boolean indicating if overlapping regions should be removed. Default is TRUE.

`verbose`
: Boolean indicating if verbose output is enabled.

Value

A k-mer table with counts for each k-mer.

`generateGenicElementCoor`
: *Function processes UCSC genePred tables to generate coordinates for various genic elements like introns, exons, CDS, UTRs, and upstream and downstream regions. It handles these coordinates with consideration for strand sensitivity and genome information.*

Description

All the operations in here are vectorized. If the table is big, expect a spike in memory. Using ncbiRefSeq table and genome hg38, the memory is stable at 4-5 GB. I can utilise data.table package to process by chunk if needed. Original table is zero-based open-end index. The indexing system is changed temporarily to follow Rs system. The output coordinate table is one-based close-end index. Critical information based on UCSC Genome website: Column Explanation bin Indexing field to speed chromosome range queries. (Only relevant to UCSC program) name Name of gene (usually transcript_id from GTF) chrom Reference sequence chromosome or scaffold strand + or - for strand txStart Transcription start position (or end position for minus strand item) txEnd Transcription end position (or start position for minus strand item) cdsStart Coding region start (or end position for minus strand item) cdsEnd Coding region end (or start position for minus strand item) exonCount Number of exons exonEnds Exon end positions (or start positions for minus strand item) exonStart Exon start positions (or end positions for minus strand item) name2 Alternate name (e.g. gene_id from GTF) cdsStartStat Status of CDS start annotation (none, unknown, incomplete, or complete) = ('none','unk','incmpl','cmpl') cdsEndStat Status of CDS end annotation (none, unknown, incomplete, or complete) exonFrames Exon frame 0,1,2, or -1 if no frame for exon (Related to codon. Number represents extra bases (modulus of 3) from previous exon block brought to a current exon block.) If cdsStart == cdsEnd, that means non-coding sequence.

- maybe cdsStartStat and cdsEndStat == "none" mean the same thing. maybe exonFrames == "-1," means the same thing.

Usage

```
generateGenicElementCoor(
  genepred,
  element.names = "all",
  upstream = NULL,
  downstream = NULL,
  genome.name = NULL,
  genome = NULL,
  return.coor.obj = FALSE
)
```

Arguments

genepred	UCSC genome name (e.g., hg19, mm39).
element.names	Types of genic elements to output: "all", "intron", "exon", "CDS", or "UTR". Default is "all".
upstream	Length of upstream sequence (can overlap other genes).
downstream	Length of downstream sequence (can overlap other genes).
genome.name	UCSC genome name for trimming overflowing coordinates.
genome	Genome object for coordinate resolution.
return.coor.obj	Whether to return a Coordinate object (default: FALSE).

Value

Genic element coordinates in a data.table or Coordinate object.

---

generateIntergenicCoor

Resolve and generate genic element coordinates from UCSC genePred table.

Description

Function generates intergenic coordinates from a UCSC genePred table. It allows users to specify the genePred data source, the relative position and minimum length for intergenic regions, and whether to return the results as a Coordinate object or a data.table.

Usage

```
generateIntergenicCoor(
  genepred,
  genome.name,
  igr.rel.pos = c(5000, 7500),
  igr.min.length = 150,
  return.coor.obj = FALSE
)
```

Arguments

genepred	UCSC genePred table or database name ("refseq" or "gencode").
genome.name	UCSC genome name (e.g., hg38, mm39).
igr.rel.pos	Intergenic relative position, defaults to c(5000, 7500).
igr.min.length	Minimum length for intergenic regions, default is 150.
return.coor.obj	Return results as a Coordinate object? Default FALSE.

Value

Intergenic coordinates as a data.table or Coordinate object.

getCOSMICauthURL *Get COSMIC authenticated URL.*

Description

To access the data for non-commercial usage, you must register with the COSMIC. This function fetch the authenticated URL from the public URL given by the COSMIC website.

Usage

```
getCOSMICauthURL(email, password, url)
```

Arguments

email	Email registered with COSMIC.
password	Password associated with the registered email.
url	Public URL provided by the COSMIC website for data access.

Value

Authenticated URL valid for 1-hour access to COSMIC data.

getCOSMICcancerGeneCensus *Get Cancer Gene Census (CGC) from COSMIC database.*

Description

To access the data for non-commercial usage, you must register with the COSMIC. This function fetch the latest CGC.

Usage

```
getCOSMICcancerGeneCensus(email, password)
```

Arguments

email	Email registered with COSMIC.
password	Password associated with the registered email.

Value

A `data.table` containing the Cancer Gene Census data.

`getCOSMIClatestVersion` *Function retrieves the latest version information of the COSMIC database and the associated genome version by scraping data from the COSMIC website.*

Description

Function retrieves the latest version information of the COSMIC database and the associated genome version by scraping data from the COSMIC website.

Usage

```
getCOSMIClatestVersion()
```

Value

A named vector containing the latest COSMIC version (`cosmic`) and genome version (`genome`).

`getCOSMICmutantExport` *Function downloads the latest Cosmic Mutant Export data from the COSMIC database. It requires the user to be registered with COSMIC for non-commercial use. The function constructs the URL for the latest mutant export file, authenticates the URL, and then downloads the data.*

Description

Function downloads the latest Cosmic Mutant Export data from the COSMIC database. It requires the user to be registered with COSMIC for non-commercial use. The function constructs the URL for the latest mutant export file, authenticates the URL, and then downloads the data.

Usage

```
getCOSMICmutantExport(email, password)
```

Arguments

`email`	Email registered with COSMIC for accessing data.
`password`	Password for the COSMIC account.

Value

A `data.table` containing the Cosmic Mutant Export data.

`getEnsemblData` *A generic function to get Ensembl data persistently from a URL. This is an internal function used by other getEnsemblXXX functions.*

Description

Error is handled based on their rule as set out at https://github.com/Ensembl/ensembl-rest/wiki/HTTP-Response-Codes

Usage

```
getEnsemblData(url, handle, max.attempt = 5)
```

Arguments

`url`	Pre-built Ensembl REST API URL.
`handle`	`curl` handle object configured for the Ensembl REST API.
`max.attempt`	Maximum number of attempts to fetch the data, default is 5.

Value

Parsed JSON data, which could be in the form of a data.frame or a list of lists, depending on the API response.

`getEnsemblRegionFeatures` *Get features of a given region.*

Description

Function fetches various genomic features for a specified region from the Ensembl database. It allows specifying the species, chromosome, region range, and types of features to query.

Usage

```
getEnsemblRegionFeatures(species, chromosome, start, end, features)
```

Arguments

`species`	Species name or alias (e.g., homo_sapiens, human).
`chromosome`	Chromosome name in Ensembl format (without 'chr' prefix).
`start`	Start position of the region.
`end`	End position of the region.
`features`	List of region features to retrieve from Ensembl. Valid options include "band", "gene", "transcript", "cds", "exon", "repeat", "simple", "misc", "variation", "somatic_variation", "structural_variation", "somatic_structural_variation", "constrained", "regulatory", "motif", "peak", "other_regulatory", "array_probe", "mane".

Value

A `data.table` containing the requested Ensembl features.

`getEnsemblVariantFeatures` *Get features of given variant IDs.*

Description

Function retrieves features for given variant IDs from the Ensembl database. It handles requests asynchronously in batches due to server limits and includes options to fetch additional variant information. Error handling for different HTTP response statuses is implemented to manage request failures.

Usage

```
getEnsemblVariantFeatures(
  species,
  variant.ids,
  include.genotypes = FALSE,
  include.phenotypes = FALSE,
  include.allele.frequencies = FALSE,
  include.genotype.frequencies = FALSE,
  curl.max.con = 100,
  verbose = 1
)
```

Arguments

`species`	Species name or alias (e.g., homo_sapiens, human).
`variant.ids`	A vector of variant IDs (e.g., rs56116432, COSM476).
`include.genotypes`	Include genotypes in the response? Default FALSE.
`include.phenotypes`	Include phenotypes in the response? Default FALSE.
`include.allele.frequencies`	Include allele frequencies? Default FALSE.
`include.genotype.frequencies`	Include genotype frequencies? Default FALSE.
`curl.max.con`	Maximum number of concurrent connections for curl requests. Default is 100.
`verbose`	Verbosity level: 1 for error only, 2 for all requests. Default 1.

Value

A variant-named list containing lists of variant features.

`getEnsemblVariantFeatures_serial` *Get features of given variant IDs.*

Description

Function fetches variant features from the Ensembl database for a set of variant IDs. It handles variant IDs in batches to comply with server limits and can include additional information like genotypes, phenotypes, allele frequencies, and genotype frequencies.

Usage

```
getEnsemblVariantFeatures_serial(
  species,
  variant.ids,
  include.genotypes = FALSE,
  include.phenotypes = FALSE,
  include.allele.frequencies = FALSE,
  include.genotype.frequencies = FALSE
)
```

Arguments

`species`	Species name or alias (e.g., homo_sapiens, human).
`variant.ids`	A vector of variant IDs (e.g., rs56116432, COSM476).
`include.genotypes`	Include genotypes in the response? Default FALSE.
`include.phenotypes`	Include phenotypes in the response? Default FALSE.
`include.allele.frequencies`	Include allele frequencies? Default FALSE.
`include.genotype.frequencies`	Include genotype frequencies? Default FALSE.

Value

A list, named by variant IDs, containing lists of variant features.

`getGnomADvariants` *Get gnomAD VCF file using tabix.*

Description

Function retrieves variant data from gnomAD VCF files using tabix for a specified set of genomic regions. It allows users to select the gnomAD version and server location (Google, Amazon, or Microsoft) for fetching the data.

Usage

```
getGnomADvariants(
  chr.names,
  starts,
  ends,
  INFO.filter = NULL,
  version = "3.1.2",
  server = "random"
)
```

Arguments

chr.names	Chromosome names.
starts	Start positions.
ends	End positions.
INFO.filter	Parse only filtered INFO ID. Default is to parse all IDs.
version	The gnomAD version. Default to latest version 3.1.2.
server	Server locations: "google", "amazon", or "microsoft". Default is random.

Value

A data.table of VCF.

---

getICTVvirusMetadataResource

Get Virus Metadata Resource (VMR) from International Committee on Taxonomy of Viruses (ICTV)

Description

Always get the latest VMR table, so no argument.

Usage

```
getICTVvirusMetadataResource()
```

Value

Virus Metadata Resource data.table.

getNCBIassemblySummary *Get NCBI assembly summary.*

Description

Retrieves the assembly summary from NCBI for a specified taxonomic group. This function allows users to obtain genome assembly information from either RefSeq or GenBank databases for various taxonomic groups.

Usage

```
getNCBIassemblySummary(organism.group, db = "refseq")
```

Arguments

organism.group	A string specifying the taxonomic group for which the assembly summary is requested. Options include 'archaea', 'bacteria', 'fungi', 'invertebrate', 'plant', 'protozoa', 'vertebrate_mammalian', 'vertebrate_other', 'viral', or 'all'.
db	A string specifying the database to use, either 'refseq' or 'genbank'.

Value

A data.table containing the assembly summary for the specified taxonomic group.

getNCBItaxonomy *Get all taxonomy classification from NCBI*

Description

NCBI provides taxonomy for all life domains in rankedlineage.dmp file. This function download and store the file in kmeRtone_data path.

Usage

```
getNCBItaxonomy()
```

Value

A data.table of taxonomy classification.

`getScores` *Function calculates scores for k-mers based on case and control k-mer counts.*

Description

Function calculates scores for k-mers based on case and control k-mer counts.

Usage

```
getScores(case.kmers, control.kmers)
```

Arguments

`case.kmers`	A data.table containing k-mer counts in case samples.
`control.kmers`	A data.table containing k-mer counts in control samples.

Value

A data.table containing scores for each k-mer.

`getUCSCgenePredTable` *Retrieve Gene Prediction Table from UCSC for a Given Genome*

Description

This function retrieves the gene prediction table from the UCSC genome database for a specified genome. It can fetch data from either the RefSeq or GENCODE databases.

Usage

```
getUCSCgenePredTable(genome.name, db)
```

Arguments

`genome.name`	A string specifying the UCSC genome name for which the gene prediction table is to be retrieved, e.g., 'hg38', 'mm39'.
`db`	A string specifying the database used by UCSC to generate the table. Options are 'refseq' or 'gencode'.

Value

A `data.table` containing the gene prediction table from the specified UCSC genome and database.

Examples

```
## Not run:
   Example usage:
   Retrieve the RefSeq gene prediction table for the human genome (hg38)
   genePredTable <- getUCSCgenePredTable(genome.name = "hg38", db = "refseq")

## End(Not run)
```

`getVCFmetainfo` *Read VCF metainfo file using tabix.*

Description

Require tabix in PATH VCF manual is referred from https://samtools.github.io/hts-specs/VCFv4.3.pdf

Usage

```
getVCFmetainfo(vcf.file)
```

Arguments

`vcf.file`	A path to a local or remote tabix-indexed VCF file.

Value

VCF metainfo.

`GET_OPTIMUM_K` *Function aims to determine the optimum k-mer size for genomic analysis based on the provided case data and genome. It may involve analyzing central patterns within the genomic sequences.*

Description

Function aims to determine the optimum k-mer size for genomic analysis based on the provided case data and genome. It may involve analyzing central patterns within the genomic sequences.

Usage

```
GET_OPTIMUM_K(case, genome, central.pattern)
```

Arguments

`case`	Coordinate class object or similar structure for case data.
`genome`	Genome class object or similar structure.
`central.pattern`	Central pattern of the k-mer.

Value

Optimum k-mer size for the given genomic data.

initKmerTable *Initialise k-mer table with all possible k-mers*

Description

Initialise k-mer table with the following columns: kmer, pos_strand, and neg_strand

Usage

```
initKmerTable(k, central.pattern = NULL, split.kmer = FALSE)
```

Arguments

k	K-mer size. Limit to 15 because vector size is limited to .Machine$integer.max. For 9- to 15-mer, the kmer sequence is separated to two columns (kmer_part1 and kmer_part2) to reduce memory significantly.
central.pattern	Central pattern(s) of the k-mer. Default is NULL.
split.kmer	Whether to split the k-mer sequence into two parts for large k values. Default is FALSE.

Value

data.table with 3 columns: kmer, pos_strand, neg_strand

kmeRtone *kmeRtone all-in-one user interface*

Description

This function serves as an all-in-one interface for various genomic data analyses leveraging k-mer based techniques.

Usage

```
kmeRtone(
  case.coor.path,
  genome.name,
  strand.sensitive,
  k,
  ctrl.rel.pos = c(80, 500),
  case.pattern,
  output.dir = "output/",
  case,
  genome,
  control,
  control.path,
  genome.path,
  rm.case.kmer.overlaps,
```

```
  single.case.len,
  merge.replicates,
  kmer.table,
  module = "score",
  rm.dup = TRUE,
  case.coor.1st.idx = 1,
  ctrl.coor.1st.idx = 1,
  coor.load.limit = 1,
  genome.load.limit = 1,
  genome.fasta.style = "UCSC",
  genome.ncbi.db = "refseq",
  use.UCSC.chr.name = FALSE,
  verbose = TRUE,
  kmer.cutoff = 5,
  selected.extremophiles,
  other.extremophiles,
  cosmic.username,
  cosmic.password,
  tumour.type.regex = NULL,
  tumour.type.exact = NULL,
  cell.type = "somatic",
  genic.elements.counts.dt,
  population.size = 1e+06,
  selected.genes,
  add.to.existing.population = FALSE,
  population.snv.dt = NULL,
  pop.plot = TRUE,
  pop.loop.chr = FALSE
)
```

Arguments

case.coor.path	Path to a folder containing chromosome-separated coordinate files or bedfiles. Assumed replicates for subfolder or bedfiles.
genome.name	Name of the genome (e.g., "hg19", "hg38"). Default is "unknown".
strand.sensitive	Logical value indicating whether strand polarity matters. Default is TRUE.
k	Length of k-mer to be investigated. Recommended values are 7 or 8.
ctrl.rel.pos	A vector of two integers specifying the relative range positions of control regions.
case.pattern	Regular expression pattern for identifying case regions. Default is NULL.
output.dir	Directory path for saving output files. Default is "output/".
case	Optional pre-built Coordinate object.
genome	Optional pre-built Genome object.
control	Optional pre-built control Coordinate object.
control.path	Path for pre-built control Coordinate object.
genome.path	Path to a directory of user-provided genome FASTA files.
rm.case.kmer.overlaps	Logical indicating whether to remove overlapping k-mers in case regions. Default is FALSE.

`single.case.len`
: Integer indicating uniform length of case regions.

`merge.replicates`
: Logical indicating whether to merge replicates. Default is TRUE.

`kmer.table`
: Pre-calculated k-mer score table.

`module`
: Selected kmeRtone module to run. Possible values include "score", "explore", "tune", among others.

`rm.dup`
: Logical indicating whether to remove duplicate coordinates. Default is TRUE.

`case.coor.1st.idx`
: Integer specifying indexing format for case coordinates.

`ctrl.coor.1st.idx`
: Integer specifying indexing format for control coordinates.

`coor.load.limit`
: Maximum number of coordinates to load. Default is 1.

`genome.load.limit`
: Maximum number of genome sequences to load. Default is 1.

`genome.fasta.style`
: String specifying the style of the genome FASTA. Possible values are "UCSC", "NCBI". Default is "UCSC".

`genome.ncbi.db`
: String specifying the NCBI database to use. Possible values are "refseq", "genbank". Default is "refseq".

`use.UCSC.chr.name`
: Logical indicating whether to use UCSC chromosome names.

`verbose`
: Logical indicating whether to display progress messages. Default is TRUE.

`kmer.cutoff`
: Cutoff percentage for k-mer selection in case studies. Default is 5.

`selected.extremophiles`
: Vector of selected extremophile species for study.

`other.extremophiles`
: Vector of other extremophile species for control.

`cosmic.username`
: COSMIC username for accessing the cancer gene census.

`cosmic.password`
: COSMIC password for accessing the cancer gene census.

`tumour.type.regex`
: Regular expression pattern for filtering tumour types.

`tumour.type.exact`
: Exact tumour type to be included in the cancer gene census.

`cell.type`
: Cell type to be included in the cancer gene census. Default is "somatic".

`genic.elements.counts.dt`
: Data table of susceptible k-mer counts in genic elements.

`population.size`
: Size of the population for cross-population studies. Default is 1 million.

`selected.genes`
: Selected genes for mutation in cross-population studies.

`add.to.existing.population`
: Logical indicating whether to add to the existing simulated population. Default is FALSE.

`population.snv.dt`
: Data table of single nucleotide variants used in population simulations.

`pop.plot`
: Logical indicating whether to plot the outcome of the cross-population study. Default is TRUE.

`pop.loop.chr`
: Logical indicating whether to loop based on chromosome name in cross-population studies. Default is FALSE.

Value

Depends on the selected module.

`Kmer_Table` *A R6 class wrapper for data.table*

Description

A R6 class wrapper for data.table

A R6 class wrapper for data.table

Details

A way to grow data.table in different environment but still retaining access to it. A temporary fix until data.table developer develop update row by reference.

Public fields

`DT` data.table of k-mers

Methods

Public methods:

- `Kmer_Table$new()`
- `Kmer_Table$print()`
- `Kmer_Table$remove_N()`
- `Kmer_Table$filter_central_pattern()`
- `Kmer_Table$update_count()`
- `Kmer_Table$update_row()`
- `Kmer_Table$clone()`

Method `new()`**:** initialize empty data.table of k-mers

Usage:

```
Kmer_Table$new()
```

Method `print()`**:** Print method.

Usage:

```
Kmer_Table$print()
```

Method `remove_N()`**:** Remove unknown base N.

Usage:

```
Kmer_Table$remove_N()
```

Method `filter_central_pattern()`**:** Filter out k-mers without defined central patterns.

Usage:

```
Kmer_Table$filter_central_pattern(central.pattern, k)
```

Arguments:

`central.pattern` Central pattern.

`k` Length of k-mer.

Returns: None.

Method `update_count()`**:** Update count for existed k-mers in the table.

Usage:

```
Kmer_Table$update_count(kmers, is.strand.sensitive, strand)
```

Arguments:

`kmers` K-mer table with new count to be added to the main table.

`is.strand.sensitive` Does strand polarity matter?

`strand` If yes, what is the strand refers to? "+" or "-".

Returns: None.

Method `update_row()`**:** Add new rows for new k-mers with their respective counts that is not existed yet in the main table.

Usage:

```
Kmer_Table$update_row(kmers, is.strand.sensitive, strand)
```

Arguments:

`kmers` K-mer table with new k-mers to be added to the main table.

`is.strand.sensitive` Does strand polarity matter?

`strand` If yes, what is the strand refers to? "+" or "-".

Returns: None.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Kmer_Table$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`loadCoordinate` *Build Coordinate object.*

Description

The Coordinate object is capable of loading genomic coordinates on demand. Chromosome-specific coordinates can be called in a bracket. The coordinates can also be expanded to k-mer size equally on both flanks

Usage

```
loadCoordinate(
  root.path = NULL,
  single.len = NULL,
  is.strand.sensitive = TRUE,
  merge.replicates = TRUE,
  rm.dup = TRUE,
  add.col.rep = FALSE,
  is.kmer = FALSE,
  k = NA,
  ori.first.index = 1,
  load.limit = 1
)
```

Arguments

root.path	A path to a directory containing either: (1) chromosome-separated coordinate files (multiple replicates is assumed for sub-folder) or (2) bedfile (multiple replicates is assumed for separate bedfiles).
single.len	Single case length relevant when all coordinates have the same length. This is for memory optimization. Default is NULL.
is.strand.sensitive	A boolean whether strand polarity matters. Default is TRUE.
merge.replicates	Merge coordinate from different replicates. Default is TRUE. If not merging, duplicates will give weight to the k-mer counting. If add.col.rep, merged coordinate will contain column replicate e.g. "rep1&rep2".
rm.dup	Remove duplicates in each replicate. Default is TRUE.
add.col.rep	Add column replicate to the coordinate table.
is.kmer	Is the coordinate refers to k-mer i.e. expanded case? Default is FALSE.
k	Length of k-mer relevant only when is.kmer is TRUE.
ori.first.index	Indexing format of the coordinate: 0 for zero-based (start, end) and 1 for one-based (start, end). Default is 1.
load.limit	Maximum number of coordinate data.table loaded on RAM. Default is 1.

Value

Coordinate object.

---

loadGenome *Build Genome object.*

Description

The Genome object is capable of loading chromosome sequence on demand. UCSC Genomes are included in this kmeRtone package. Their specific chromosome sequence will be downloaded on demand once.

Usage

```
loadGenome(
  genome.name,
  fasta.style,
  mask = "none",
  fasta.path,
  ncbi.db,
  ncbi.asm,
  use.UCSC.name = FALSE,
  load.limit = 1
)
```

Arguments

genome.name	A genome name. UCSC and NCBI genome is included with kmeRtone. Input their name e.g. hg19 or GRCh37.
fasta.style	FASTA version: "UCSC" or "NCBI".
mask	Genome mask: "none", "soft", or "hard". Default is "none".
fasta.path	Path to the fasta file as a character vector.
ncbi.db	NCBI database: "refseq" or "genbank".
ncbi.asm	NCBI assembly table.
use.UCSC.name	For NCBI Genome, use UCSC-style chromosome name? Default is FALSE.
load.limit	Maximum chromosome sequences loaded. Default is 1.

Value

A UCSC_Genome or NCBI_Genome object.

Examples

```
## Not run:
   genome <- loadGenome(...)
   genome["chr1"]
   genome[c("chr1", "chr2")]

## End(Not run)
```

---

loadGenomicContents *Function calculates various genomic content metrics based on the provided genome object.*

Description

Function calculates various genomic content metrics based on the provided genome object.

Usage

```
loadGenomicContents(genome)
```

Arguments

genome	An object of class 'NCBI_Genome' containing genomic information.

Value

A data.table containing calculated genomic content metrics.

---

mapKmers *Map k-mers of a given sequence and coordinate*

Description

This function maps k-mers within a specified sequence based on provided start and end coordinates, or based on a fixed length.

Usage

```
mapKmers(seq, start, end = NULL, len = NULL, k, rm.trunc.kmer = TRUE)
```

Arguments

seq	A single sequence string in which k-mers are to be mapped.
start	A vector of start coordinates for mapping k-mers. If only start positions are provided, exact k-mer extraction is performed.
end	A vector of end coordinates corresponding to the start positions. If NULL, all regions are assumed to have the same length. Used for varied region lengths to perform a sliding window.
len	An integer specifying the fixed length of regions. Used when regions have a uniform length greater than k. End coordinates are assumed NULL in this case.
k	An integer specifying the length of k-mers to be mapped.
rm.trunc.kmer	Logical indicating whether to remove truncated k-mers resulting from out-of-bound regions. Default is TRUE.

Value

A vector of mapped k-mers.

mergeCoordinate *Merge overlapping or continuous regions.*

Description

Table must have start and end columns. The output is exactly similar to the reduce function from GenomicRanges.

Usage

```
mergeCoordinate(coor)
```

Arguments

coor	Coordinate data.table.

Value

Merged coordinate data.table.

mixColors *Mix color*

Description

This is useful to get overlayed colors.

Usage

```
mixColors(cols, alpha)
```

Arguments

cols	Colors in hex format or R color code e.g. "red", "black", etc.
alpha	Add alpha transparency value.

Value

New mixed colors in hex format.

NCBI_Genome *Class constructor - build NCBI Genome object*

Description

Class constructor - build NCBI Genome object

Class constructor - build NCBI Genome object

Details

NCBI FASTA file contain nucleotide accession number at the headers, followed by some information about the sequence whether they are chromosome, plasmid, or mictochondria, their assembly status, etc.

Public fields

`fasta_file` A path to FASTA file. fasta files.

`genome_name` A genome name.

`db` NCBI database: "refseq" or "genbank"

`seq` A chromosome-named list of sequences.

`seq_len` A chromosome-named vector of sequence length.

`load_limit` Maximum chromosome sequences loaded.

`mask` Genome mask status: "hard", "soft", or "none".

`use_UCSC_name` Use UCSC style chromosome name? Default to FALSE.

`headers` A chromosome-named vector of headers.

`avail_seqs` Available chromosome sequences in the fasta file.

`asm` Assembly summary.

Methods

Public methods:

- `NCBI_Genome$new()`
- `NCBI_Genome$[()`
- `NCBI_Genome$print()`
- `NCBI_Genome$get_assembly_report()`
- `NCBI_Genome$clone()`

Method `new()`**:** Create a new NCBI Genome class

Usage:

```
NCBI_Genome$new(
  genome.name,
  db,
  fasta.file,
  asm,
  mask,
  use.UCSC.name,
  load.limit
)
```

Arguments:

`genome.name` A genome name. NCBI genome is included with kmeRtone.

`db` NCBI database: "refseq" or "genbank".

`fasta.file` A path to the NCBI-style fasta files. This is for user's own FASTA file.

`asm` NCBI assembly summary.

`mask` Genome mask status: "hard", "soft", or "none". Default is "none".

`use.UCSC.name` Use UCSC style chromosome name? Default to FALSE.

`load.limit` Maximum chromosome sequences loaded. Default is 1.

Returns: A new `NCBI Genome` object.

Method `[()`: Calling chromosome sequence by loading on demand. Maximum load is determine by load_limit field.

Usage:

```
NCBI_Genome$[(chr.names, reload = FALSE)
```

Arguments:

`chr.names` Chromosome name. It can be a vector of chromosomes.

`reload` Reload the sequence from the fasta_file. Default is FALSE.

Returns: A single or list of sequence of requested chromosome.

Method `print()`: Print summary of `Genome` object.

Usage:

```
NCBI_Genome$print()
```

Returns: Message of `Genome` object summary.

Method `get_assembly_report()`: Get NCBI assembly report for the genome.

Usage:

```
NCBI_Genome$get_assembly_report()
```

Returns: Message of `Genome` object summary.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
NCBI_Genome$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`partitionCoordinate` *Partition overlapping or continuous regions.*

Description

Table must have start and end columns. The mechanism is similar to the disjoin function from GenomicRanges but the end coordinate is different.

Usage

```
partitionCoordinate(coor)
```

Arguments

coor	Coordinate data.table.

Value

Partitioned coordinate data.table.

---

persistentDownload *Download file until successful*

Description

If download failed, it will be repeated until max attempt reached.

Usage

```
persistentDownload(
  file.url,
  output.name,
  max.attempt = 5,
  user.invoke = TRUE,
  header
)
```

Arguments

file.url	File uniform resource locator.
output.name	Output name.
max.attempt	Maximum number of attempt. Default is 5.
user.invoke	If number of attempt is reached, ask user whether to keep trying. Default is TRUE to invoke the prompt.
header	A named list or vector of curl header.

Value

A downloaded file.

readBCF *Read BCF files and extract chromosome information*

Description

This function reads BCF (Binary Variant Call Format) files and extracts chromosome information.

Usage

```
readBCF(fname, regions, is_file)
```

Arguments

fname	The file name or URL of the BCF file.
regions	A string specifying the regions to be read from the BCF file.
is_file	Boolean indicating if the regions parameter is a file.

Value

A list containing chromosome information.

readBED *Read a BED file. One-based indexing is enforced.*

Description

Read a BED file. One-based indexing is enforced.

Usage

```
readBED(bed.path)
```

Arguments

bed.path	A path to a BED file.

Value

data.table.

readFASTA *Read FASTA files.*

Description

Read FASTA files.

Usage

```
readFASTA(fasta.file)
```

Arguments

fasta.file	A path to a FASTA file.

Value

A sequence vector with header names

readSingleFASTA *Read fast single-header FASTA file.*

Description

Read fast single-header FASTA file.

Usage

```
readSingleFASTA(file_path, mask = "none")
```

Arguments

file_path	A path to FASTA file.
mask	Capitalize all base letters?

Value

A single sequence string without header.

readVCF — *Read VCF file using tabix.*

Description

Require tabix in PATH VCF manual is referred from https://samtools.github.io/hts-specs/VCFv4.3.pdf

Usage

```
readVCF(vcf.file, chr.names, starts, ends, INFO.filter = NULL)
```

Arguments

vcf.file	A path to a local or remote tabix-indexed VCF file.
chr.names	Chromosome names.
starts	Start positions.
ends	End positions.
INFO.filter	Parse only filtered INFO ID. Default is to parse all IDs.

Value

A data.table of VCF.

readVCF2 — *Read VCF file using tabix.*

Description

Require tabix in PATH VCF manual is referred from https://samtools.github.io/hts-specs/VCFv4.3.pdf

Usage

```
readVCF2(vcf.file, chr.names, starts, ends, INFO.filter = NULL)
```

Arguments

vcf.file	A path to a local or remote tabix-indexed VCF file.
chr.names	Chromosome names.
starts	Start positions.
ends	End positions.
INFO.filter	Parse only filtered INFO ID. Default is to parse all IDs.

Value

A data.table of VCF.

removeRegion *Remove overlapping region in coordinate* data.table.

Description

Any "coor" that overlap within the "region" will be removed e.g. region = 10-20 and coor = 1-30 The results will be: coor = 1-10, 20-30 The coor still overlap one base at the terminal. This is done to produce exact result as the previous MPhil research.

Usage

```
removeRegion(coor, region)
```

Arguments

coor	Coordinate data.table.
region	A data.table of region coordinate to be removed.

Value

New coordinate data.table with the regions removed.

reverseComplement *Get reverse complement sequence of DNA*

Description

Get reverse complement sequence of DNA

Usage

```
reverseComplement(DNA.sequence, form = "string")
```

Arguments

DNA.sequence	DNA sequence can be in a form of character vector or string. Multiple sequences are accepted.
form	Specify the form: "string" of "vector". Default is "string"

Value

Reverse complementary sequence

Examples

```
## Not run:
   reverseComplement("AAAAA")
   reverseComplement(c("AAAAA", "CCCCC"))
   reverseComplement(c("A", "A", "A", "A"), form = "vector")

## End(Not run)
```

SCORE *Calculate susceptibility scores for k-mers in case and control regions.*

Description

Function calculates susceptibility scores for k-mers in case and control regions. Case regions are defined by genomic coordinates provided in a file or data.table. Control regions can be constructed relative to the case regions or provided directly. The scores are computed based on the occurrence of k-mers in case and control regions.

Usage

```
SCORE(
  case.coor.path,
  genome.name,
  strand.sensitive,
  k,
  ctrl.rel.pos,
  case.pattern,
  output.path,
  case,
  genome,
  control,
  control.path,
  genome.path,
  rm.case.kmer.overlaps,
  single.case.len,
  merge.replicates,
  rm.dup,
  case.coor.1st.idx,
  ctrl.coor.1st.idx,
  coor.load.limit,
  genome.load.limit,
  genome.fasta.style,
  genome.ncbi.db,
  use.UCSC.chr.name,
  verbose
)
```

Arguments

case.coor.path	Path to the file containing genomic coordinates of case regions.
genome.name	Name of the genome to be used.
strand.sensitive	Logical indicating whether strand information should be considered.
k	Length of the k-mers to be extracted.
ctrl.rel.pos	Relative positions of control regions with respect to case regions. It should be a vector of two integers indicating the upstream and downstream distances from the case regions.
case.pattern	Regular expression pattern to identify the central sequence in case regions.

output.path	Directory path where the output files will be saved.
case	Data.table containing the genomic coordinates of case regions.
genome	Genome data.table containing the genomic sequence information.
control	Data.table containing the genomic coordinates of control regions.
control.path	Path to the file containing genomic coordinates of control regions (optional).
genome.path	Path to the genome FASTA file.
rm.case.kmer.overlaps	Logical indicating whether overlapping k-mers within case regions should be removed.
single.case.len	Single case length.
merge.replicates	Logical indicating whether replicates should be merged.
rm.dup	Logical indicating whether duplicate k-mers should be removed.
case.coor.1st.idx	First index in the case coordinate file.
ctrl.coor.1st.idx	First index in the control coordinate file.
coor.load.limit	Maximum number of coordinates to load.
genome.load.limit	Maximum number of genome sequences to load.
genome.fasta.style	FASTA style.
genome.ncbi.db	NCBI database.
use.UCSC.chr.name	Logical indicating whether to use UCSC chromosome names.
verbose	Logical indicating whether to display progress messages.

Value

Data.table containing susceptibility scores for k-mers.

`scoreKmers` *Function calculates the Z-score for each k-mer based on the observed case counts and expected case counts under the null hypothesis.*

Description

Function calculates the Z-score for each k-mer based on the observed case counts and expected case counts under the null hypothesis.

Usage

```
scoreKmers(kmer.table)
```

Arguments

kmer.table	A data.table containing k-mer counts, where each row represents a k-mer and columns "case" and "control" represent the counts in case and control samples respectively.

Value

A modified version of the input kmer.table with an additional column "z" containing the calculated Z-scores for each k-mer.

---

selectGenomesForCrossSpeciesStudy

Select genomes for cross-species studies.

Description

The following filters are applied:

1. assembly_level: "Complete Genome" or "Chromosome"
2. genome_rep: "Full"
3. Unique species_taxid (single representative species)
4. refseq_category of "reference genome" is prioritised over "representative genome"

Usage

```
selectGenomesForCrossSpeciesStudy(organism.group = "bacteria", db = "refseq")
```

Arguments

organism.group	Species group: archaea, bacteria, fungi, invertebrate, plant, protozoa, vertebrate_mammalian, vertebrate_other, or viral.
db	Database record to use: refseq or genbank

Value

NCBI assembly summary with added column organism.group.

`selectRepresentativeFromASM` *Select the best representative species from the NCBI assembly summary.*

Description

sort.idx is a weight to sort where heavier will be preffered. Any tie weight will be further sorted by organism_name. Only the top unique species_taxid will be retained in the final assembly summary.

Usage

```
selectRepresentativeFromASM(asm)
```

Arguments

`asm`	NCBI assembly summary.

Value

Trimmed NCBI assembly summary.

`simulatePopulation` *Simulate a population given ranges of chromosome sequence to mutate.*

Description

Simulate a population given ranges of chromosome sequence to mutate.

Usage

```
simulatePopulation(
  chrom_seq,
  starts,
  ends,
  strand,
  snv_df,
  pop_size,
  top_kmers,
  central_pattern,
  k
)
```

Arguments

chrom_seq	A chromosome sequence.
starts	Start positions.
ends	End positions.
strand	Strand type: "+" or "-".
snv_df	A table of SNV frequency. Columns: position, base, count.
pop_size	Size of population.
top_kmers	Extreme k-mers i.e. highly susceptible k-mers.
central_pattern	K-mer central pattern.
k	K-mer size.

Value

A count matrix with 4 rows for total top k-mers and susceptible k-mers in sense and antisense. Columns correspond to population individuals.

---

splitFASTA *Split a FASTA file by header.*

Description

The first non-space word in the header will be used as the file name.

Usage

```
splitFASTA(fasta.file, output.dir = "./")
```

Arguments

fasta.file	A path to a FASTA file.
output.dir	A path to save the output results. Default is current working directory.

Details

data.table::fread is not built for reading in chunks. The developers expect skip and nrow arguments to be in a small number. Large number slows the reading a bit.

Value

A splitted fasta files by its headers.

splitFASTA2 *Read fast single-header FASTA file.*

Description

Slower than data.table::fread

Usage

```
splitFASTA2(file_path, output_dir)
```

Arguments

file_path	A path to FASTA file.
output_dir	A path to save the output results.

Value

A single sequence string without header.

STUDY_ACROSS_POPULATIONS *Study k-mer composition of selected COSMIC causal cancer genes across human populations worldwide.*

Description

Simulation of human population is based on single nucleotide variantion.

Usage

```
STUDY_ACROSS_POPULATIONS(
  kmer.table,
  kmer.cutoff = 5,
  genome.name,
  k,
  db = "refseq",
  central.pattern = NULL,
  population.size = 1e+06,
  selected.genes,
  add.to.existing.population = FALSE,
  output.dir = "study_across_populations/",
  population.snv.dt = NULL,
  loop.chr = TRUE,
  plot = FALSE
)
```

Arguments

kmer.table	A data.table of kmer table.
kmer.cutoff	Percentage of extreme kmers to study. Default to 5.
genome.name	UCSC genome name.
k	K-mer size.
db	Database used by UCSC to generate gene prediction: "refseq" or "gencode". Default is "refseq".
central.pattern	K-mer's central patterns. Default is NULL.
population.size	Size of population to simulate. Default is 1 million.
selected.genes	Set of genes to study e.g. skin cancer genes.
add.to.existing.population	Add counts to counts.csv? Default is FALSE.
output.dir	A directory for the outputs. Default to study_across_populations.
population.snv.dt	Population SNV table.
loop.chr	Loop chromosome?. Default is TRUE. If FALSE, beware of a memory spike because of VCF content. VCF contains zero counts for every population. Input pre-computed trimmed-version population.snv.dt.
plot	Boolean. Default is FALSE. If TRUE, will plot results.

Value

An output directory containing plots.

STUDY_ACROSS_SPECIES *Study k-mer composition across species.*

Description

Analysis of distribution of highly enriched k-mers across species.

Usage

```
STUDY_ACROSS_SPECIES(
  kmer.table,
  kmer.cutoff = 5,
  k,
  central.pattern = NULL,
  selected.extremophiles,
  other.extremophiles,
  output.dir = "study_across_species/"
)
```

Arguments

kmer.table	A data.table of kmer table or path to it.
kmer.cutoff	Percentage of extreme kmers to study. Default to 5 percent.
k	K-mer size.
central.pattern	K-mer's central patterns. Default is NULL.
selected.extremophiles	A vector of selected extremophile species. e.g. c("Deinococcus soli", "Deinococcus deserti") The best representative will be selected from the assembly summary.
other.extremophiles	A vector of other extremophile species. These are used as a control to compare with the selected extremophiles.
output.dir	A directory for the outputs.

Value

An output directory containing plots.

STUDY_CANCER_GENES *Study k-mer composition of causal cancer genes from COSMIC Cancer Gene Census (CGC) database.*

Description

Detail of Cancer Gene Census can be accessed and read at https://cancer.sanger.ac.uk/census

Usage

```
STUDY_CANCER_GENES(
  cosmic.username,
  cosmic.password,
  tumour.type.regex = NULL,
  tumour.type.exact = NULL,
  cell.type = "somatic",
  genic.elements.counts.dt,
  output.dir = "study_cancer_genes/"
)
```

Arguments

cosmic.username	COSMIC username i.e. registered email.
cosmic.password	COSMIC password.
tumour.type.regex	Regular expression for "Tumour Types" column in Cancer Gene Census table. Default is NULL.

`tumour.type.exact`
: Exact keywords for "Tumour Types" column in Cancer Gene Census table. Default is NULL.

`cell.type`
: Type of cell: "somatic" or "germline". Default is "somatic".

`genic.elements.counts.dt`
: Genic element count table generated from STUDY_GENIC_ELEMENTS.

`output.dir`
: A directory for the outputs.

Value

An output directory containing plots.

`STUDY_GENIC_ELEMENTS` *Study k-mer composition across species.*

Description

Study k-mer composition across species.

Usage

```
STUDY_GENIC_ELEMENTS(
  kmer.table,
  kmer.cutoff = 5,
  k,
  genome.name = "hg38",
  central.pattern = NULL,
  db = "refseq",
  output.dir = "study_genic_elements/"
)
```

Arguments

`kmer.table`
: A data.table of kmer table.

`kmer.cutoff`
: Percentage of extreme kmers to study. Default to 5.

`k`
: K-mer size.

`genome.name`
: UCSC genome name.

`central.pattern`
: K-mer's central patterns. Default is NULL.

`db`
: Database used by UCSC to generate gene prediction: "refseq" or "gencode". Default is "refseq".

`output.dir`
: A directory for the outputs.

Value

An output directory containing plots.

system3 *A system2 wrapper. If anything happen, just give me error!*

Description

Turn warning to error.

Usage

```
system3(
  command,
  args = character(),
  stdout = "",
  stderr = "",
  stdin = "",
  input = NULL,
  env = character(),
  wait = TRUE,
  minimized,
  invisible,
  timeout = 0
)
```

Arguments

command	the system command to be invoked, as a character string.
args	a character vector of arguments to command.
stdout, stderr	where output to 'stdout' or 'stderr' should be sent. Possible values are "", to the R console (the default), NULL or FALSE (discard output), TRUE (capture the output in a character vector) or a character string naming a file.
stdin	should input be diverted? "" means the default, alternatively a character string naming a file. Ignored if input is supplied.
input	if a character vector is supplied, this is copied one string per line to a temporary file, and the standard input of command is redirected to the file.
env	character vector of name=value strings to set environment variables.
wait	a logical (not NA) indicating whether the R interpreter should wait for the command to finish, or run it asynchronously. This will be ignored (and the interpreter will always wait) if stdout = TRUE or stderr = TRUE. When running the command asynchronously, no output will be displayed on the Rgui console in Windows (it will be dropped, instead).
minimized, invisible	arguments that are accepted on Windows but ignored on this platform, with a warning.
timeout	timeout in seconds, ignored if 0. This is a limit for the elapsed time running command in a separate process. Fractions of seconds are ignored.

trimCoordinate *Trim out-of-bound coordinates*

Description

It operates in two mode: coordinate table with and without chromosome. The former require `Genome` for the chromosomal sequence length.

Usage

```
trimCoordinate(coor, seq.len, genome)
```

Arguments

`coor`	Coordinate `data.table`.
`seq.len`	Sequence length to trim end position.
`genome`	`Genome` class object.

Value

Trimmed coordinate `data.table`.

UCSC_Genome *Class constructor - build Genome object*

Description

Class constructor - build Genome object

Class constructor - build Genome object

Public fields

`root_path` A path to a directory containing chromosome-separated fasta files.

`genome_name` A genome name.

`paths` Individual chromosome sequence files.

`seq` A chromosome-named list of sequences.

`seq_len` A chromosome-named vector of sequence length.

`load_limit` Maximum chromosome sequences loaded.

`mask` Genome mask status: "hard", "soft", or "none".

`info_file` Path to info file with pre-computed values.

`chr_names` Chromosome names.

Methods

Public methods:

- UCSC_Genome$new()
- UCSC_Genome$[()
- UCSC_Genome$print()
- UCSC_Genome$get_length()
- UCSC_Genome$get_content()
- UCSC_Genome$clone()

Method `new()`**:** Create a new Genome class

Usage:

```
UCSC_Genome$new(genome.name, root.path, mask, load.limit)
```

Arguments:

`genome.name` A genome name. UCSC genome is included with kmeRtone.

`root.path` A path to a directory containing chromosome-separated fasta files.

`mask` Genome mask status: "hard", "soft", or "none". Default is "none".

`load.limit` Maximum chromosome sequences loaded. Default is 1.

Returns: A new `Genome` object.

Method `[()`**:** Calling chromosome sequence by loading on demand. Maximum load is determine by load_limit field.

Usage:

```
UCSC_Genome$[(chr.names, reload = FALSE)
```

Arguments:

`chr.names` Chromosome name. It can be a vector of chromosomes.

`reload` Reload the sequence from the root_path. Default is FALSE.

Returns: A single or list of sequence of requested chromosome.

Method `print()`**:** Print summary of `Genome` object.

Usage:

```
UCSC_Genome$print()
```

Returns: Message of `Genome` object summary.

Method `get_length()`**:** Get chromosome length from pre-calculated length

Usage:

```
UCSC_Genome$get_length(chr.names, recalculate = FALSE)
```

Arguments:

`chr.names` Chromosome name. It can be a vector of chromosomes.

`recalculate` Recalculate the pre-calculated length.

Returns: A chromosome-named vector of length value.

Method `get_content()`**:** Get pre-calculated sequence content e.g. G+C content

Usage:

```
UCSC_Genome$get_content(chr.names, seq, recalculate = FALSE)
```

Arguments:

`chr.names` Chromosome name. It can be a vector of chromosomes.
`seq` Sequence to count. e.g. c("G", "C")
`recalculate` Recalculate the pre-calculated length.
Returns: A chromosome-named vector of sequence content.

Method `clone()`**:** The objects of this class are cloneable with this method.
Usage:
```
UCSC_Genome$clone(deep = FALSE)
```
Arguments:
`deep` Whether to make a deep clone.

`writeBED` *Write a BED file. Zero-based indexing is enforced.*

Description

Write a BED file. Zero-based indexing is enforced.

Usage

```
writeBED(bed, output.filename)
```

Arguments

`bed`	A BED `data.table`.
`output.filename`	An output BED filename.

`writeFASTA` *Write FASTA files.*

Description

Write FASTA files.

Usage

```
writeFASTA(seqs, fasta.path, append = FALSE)
```

Arguments

`seqs`	A vector or list of sequences with header name. If it is a list, it must only contain one single sequence string for every element e.g. list(chr1 = "NNNNNNNN") not list(chr1 = c("NNNNNN", "AAAAAA"))
`fasta.path`	A path to a FASTA file.
`append`	Boolean. Default is FALSE. If TRUE, will append the results to existing file.

Value

None

writeVCF *Write VCF file and compress using bgzip.*

Description

Require bgzip in PATH VCF manual is referred from https://samtools.github.io/hts-specs/VCFv4.3.pdf

Usage

```
writeVCF(vcf, output.vcf.gz, append = FALSE, tabix = FALSE)
```

Arguments

vcf	A VCF object.
output.vcf.gz	Output filename including vcf.gz extension.
append	To append or not? Default is FALSE.
tabix	To tabix or not? Default is FALSE.

Index

∗ **datasets**
example_genome_coor, 21
example_kmeRtone_score, 22

addAlphaCol, 4

bedToCoor, 4
buildControl, 5
buildKmerTable, 5
buildMidPatternKmerTable, 6
buildRangedKmerTable, 6
buildRESTurl, 7

calKmerSkew, 8
calPWM, 8
catHeader, 9
Coordinate, 9
count_substring_fixed, 19
count_substring_regex, 20
countBaseComposition, 11
countChoppedKmers, 12
countDistribution, 13
countKmers, 13
countMidPatternContext, 14
countMidPatternContext2, 14
countMidPatternKmers, 15
countMidPatternRangedKmers, 16
countPointContext, 16
countPointContext2, 17
countRangedKmers, 17
countRevCompKmers, 18
countSlidingWindow, 18
countSlidingWindow2, 19

downloadNCBIGenomes, 20
downloadUCSCgenome, 21

example_genome_coor, 21
example_kmeRtone_score, 22
EXPLORE, 23
extractKmers, 24

generateGenicElementCoor, 25
generateIntergenicCoor, 26
GET_OPTIMUM_K, 35
getCOSMICauthURL, 27
getCOSMICcancerGeneCensus, 27
getCOSMIClatestVersion, 28
getCOSMICmutantExport, 28
getEnsemblData, 29
getEnsemblRegionFeatures, 29
getEnsemblVariantFeatures, 30
getEnsemblVariantFeatures_serial, 31
getGnomADvariants, 31
getICTVvirusMetadataResource, 32
getNCBIassemblySummary, 33
getNCBItaxonomy, 33
getScores, 34
getUCSCgenePredTable, 34
getVCFmetainfo, 35

initKmerTable, 36

Kmer_Table, 39
kmeRtone, 36

loadCoordinate, 40
loadGenome, 41
loadGenomicContents, 42

mapKmers, 43
mergeCoordinate, 44
mixColors, 44

NCBI_Genome, 45

partitionCoordinate, 46
persistentDownload, 47

readBCF, 48
readBED, 48
readFASTA, 49
readSingleFASTA, 49
readVCF, 50
readVCF2, 50
removeRegion, 51
reverseComplement, 51

SCORE, 52
scoreKmers, 53

selectGenomesForCrossSpeciesStudy, 54
selectRepresentativeFromASM, 55
simulatePopulation, 55
splitFASTA, 56
splitFASTA2, 57
STUDY_ACROSS_POPULATIONS, 57
STUDY_ACROSS_SPECIES, 58
STUDY_CANCER_GENES, 59
STUDY_GENIC_ELEMENTS, 60
system3, 61

trimCoordinate, 62

UCSC_Genome, 62

writeBED, 64
writeFASTA, 64
writeVCF, 65