

Package ‘single’

December 1, 2022

Type Package

Title Accurate consensus sequence from nanopore reads of a gene library

Version 1.2.0

biocViews Software, Sequencing

Depends R (>= 4.1)

Description Accurate consensus sequence from nanopore reads of a DNA gene library. SINGLE corrects for systematic errors in nanopore sequencing reads of gene libraries and it retrieves true consensus sequences of variants identified by a barcode, needing only a few reads per variant. More information in preprint doi: <https://doi.org/10.1101/2020.03.25.007146>.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Imports Biostrings, BiocGenerics, dplyr, GenomicAlignments, IRanges, methods, reshape2, rlang, Rsamtools, stats, stringr, tidyr, utils

Suggests BiocStyle, knitr, rmarkdown

VignetteBuilder knitr

RoxygenNote 7.1.2

git_url <https://git.bioconductor.org/packages/single>

git_branch RELEASE_3_16

git_last_commit 30e7ba3

git_last_commit_date 2022-11-01

Date/Publication 2022-12-01

Author Rocio Espada [aut, cre] (<<https://orcid.org/0000-0003-3829-473X>>)

Maintainer Rocio Espada <rocio.espada@espci.fr>

R topics documented:

| | |
|--------------------------------------|----|
| bases | 2 |
| evaluate_fits | 2 |
| fit_logregr | 3 |
| glm.predict. | 5 |
| list_mismatches | 5 |
| mutation_rate | 6 |
| pileup_by_QUAL | 6 |
| p_prior_errors | 7 |
| p_prior_mutations | 8 |
| single_consensus_byBarcode | 9 |
| single_evaluate | 10 |
| single_train | 11 |
| weighted_consensus | 13 |

| | |
|--------------|-----------|
| Index | 14 |
|--------------|-----------|

| | |
|-------|--------------|
| bases | <i>Bases</i> |
|-------|--------------|

Description

Vector A C G T -

Usage

bases

Format

Character vector of length 5

| | |
|---------------|-----------------------------|
| evaluate_fits | <i>Evaluate SINGLE fits</i> |
|---------------|-----------------------------|

Description

Evaluates SINGLE for pos, nucleotides and QUAL in the given ranges.

Usage

```
evaluate_fits(
  pos_range,
  q_range,
  output_file,
  data_fits,
  ref_seq,
  verbose = FALSE,
  save = FALSE
)
```

Arguments

| | |
|-------------|--|
| pos_range | Numeric vector. Positions to evaluate. |
| q_range | Numeric vector. QUAL to evaluate. |
| output_file | File name for output, if save=TRUE. |
| data_fits | Data.frame with columns position nucleotide slope intercept as the one returned by fit_logregr |
| ref_seq | DNAStringSet containing the true reference sequence. |
| verbose | Logical. |
| save | Logical. Should results be saved in output_file? |

Value

data.frame with SINGLE fits evaluated for pos_range and q_range.

Examples

```
refseq_fasta = system.file("extdata", "ref_seq.fasta", package = "single")
ref_seq = Biostrings::readDNAStringSet(refseq_fasta)
fits_file <- system.file("extdata", "fits_example.txt", package = "single")
fits <- read.table(fits_file, header=TRUE)
pos_range = seq_len(100)
q_range = seq(1,50)
evaluated_fits <- evaluate_fits(pos_range = c(1,5), q_range = c(0,10),
                               data_fits = fits, ref_seq = ref_seq)
```

fit_logregr

Fit SINGLE's logistic regression

Description

This is an auxiliary function in single package. It takes counts_pnq and for each position and nucleotide it fits SINGLE's logistic regression.

Usage

```
fit_logregr(
  counts_pnq,
  ref_seq,
  p_prior_errors,
  p_prior_mutations,
  save = FALSE,
  output_file_fits,
  output_file_data,
  verbose = FALSE
)
```

Arguments

| | |
|-------------------|---|
| counts_pnq | Data frame with columns position nucleotide quality counts, as returned by <code>pileup_by_QUAL</code> |
| ref_seq | DNAStringSet containing the true reference sequence. |
| p_prior_errors | Data frame with columns position nucleotide prior.error, as the one returned by <code>p_prior_errors()</code> . |
| p_prior_mutations | Data frame with columns wt.base, nucleotide and p_mutation (probability of mutation), as the one returned by <code>p_prior_mutations()</code> . |
| save | Logical. Should data be saved in a output_file? |
| output_file_fits | File into which save the single fits if save=TRUE |
| output_file_data | File into which save the fitted data if save=TRUE |
| verbose | Logical. |

Value

data.frame with columns position, nucleotide, slope and intercept (of the sigmoidal regression).

Examples

```
refseq_fasta <- system.file("extdata", "ref_seq.fasta", package = "single")
ref_seq = Biostrings::readDNAStringSet(refseq_fasta)
train_reads_example <- system.file("extdata", "train_seqs_500.sorted.bam",
  package = "single")
counts_pnq <- pileup_by_QUAL(bam_file=train_reads_example,
  pos_start=1, pos_end=10)
p_prior_mutations <- p_prior_mutations(rates.matrix = mutation_rate,
  mean.n.mut = 5, ref_seq = ref_seq)
p_prior_errors <- p_prior_errors(counts_pnq=counts_pnq)
fits <- fit_logregr(counts_pnq = counts_pnq, ref_seq=ref_seq,
  p_prior_errors = p_prior_errors, p_prior_mutations = p_prior_mutations)
```

glm.predict. *Computes prior probability of mutations*

Description

This is an auxiliary function in single package. It evaluates the sigmoidal function given by the parameters slope and intercept on x.

Usage

```
glm.predict.(x, slope, intercept)
```

Arguments

| | |
|-----------|--|
| x | Numeric. Values to evaluate. |
| slope | Slope of the sigmoidal function to evaluate. |
| intercept | Intercept of the sigmoidal function to evaluate. |

Value

Numeric.

Examples

```
x = c(-10:10)
y = glm.predict.(x,1,2)
plot(x,y)
```

list_mismatches *Lists mismatches between two DNAstrings*

Description

This is an auxiliary function in single package, to list the mutations of two DNAstrings.

Usage

```
list_mismatches(ref, seq)
```

Arguments

| | |
|-----|--|
| ref | DNAStrng, reference sequence. |
| seq | DNAStrng, target sequence, same length as ref. |

Value

Character vector containing Nucleotide in ref Position Nucleotide in seq. If ref and seq are equal, it returns NA.

Examples

```
ref = Biostrings::DNASTring("AAAA")
seq = Biostrings::DNASTring("AGAT")
list_mismatches(ref, seq)
list_mismatches(ref, ref)
```

| | |
|---------------|----------------------|
| mutation_rate | <i>mutation_rate</i> |
|---------------|----------------------|

Description

Mutational rate matrix for error-prone PCR, obtained from GeneMorph II Random Mutagenesis Kit.

Usage

```
mutation_rate
```

Format

```
matrix size 4x5
```

Source

<https://www.agilent.com/cs/library/usermanuals/public/200550.pdf>

| | |
|----------------|-----------------------|
| pileup_by_QUAL | <i>Pileup by QUAL</i> |
|----------------|-----------------------|

Description

To explain

Usage

```
pileup_by_QUAL(
  bam_file,
  QUAL_values = seq(93, 0),
  pos_start = NA,
  pos_end = NA
)
```

Arguments

| | |
|-------------|--|
| bam_file | Bam file to pile up |
| QUAL_values | Numeric vector. QUAL values to analyze in the data. |
| pos_start | Numeric. Position to start analyzing, counting starts from 1 and it refers to reference used for minimap2 alignment. |
| pos_end | Numeric. Position to stop analyzing, counting starts from 1 and it refers to reference used for minimap2 alignment. |

Value

data.frame with columns strand,pos,nucleotide,QUAL,countss

Examples

```
refseq_fasta <- system.file("extdata", "ref_seq.fasta", package = "single")
train_reads_example <- system.file("extdata", "train_seqs_500.sorted.bam",
                                   package = "single")
counts_pnq <- pileup_by_QUAL(bam_file=train_reads_example,
                             pos_start=1,pos_end=10)
head(counts_pnq)
```

p_prior_errors *Computes prior probability of errors*

Description

This is an auxiliary function in single package. It takes a data frame with counts by position, nucleotide and Qscore and it summarises it into proportion of nucleotide counts by position.

Usage

```
p_prior_errors(counts_pnq, output_file = NULL, save = FALSE)
```

Arguments

| | |
|-------------|---|
| counts_pnq | Data frame with columns position nucleotide quality counts, as returned by parse_counts_pnq |
| output_file | File name for output, if save=TRUE. |
| save | Logical. Should data be saved in a output_file? |

Value

Data frame with columns position nucleotide prior.error.

Examples

```
refseq_fasta <- system.file("extdata", "ref_seq.fasta", package = "single")
train_reads_example <- system.file("extdata", "train_seqs_500.sorted.bam",
                                   package = "single")
counts_pnq <- pileup_by_QUAL(train_reads_example, pos_start=1, pos_end=10)
p_prior_errors <- p_prior_errors(counts_pnq=counts_pnq)
head(p_prior_errors)
```

| | |
|-------------------|--|
| p_prior_mutations | <i>Computes prior probability of mutations</i> |
|-------------------|--|

Description

This is an auxiliary function in single package. It computes the prior probability of mutation in a gene library.

Usage

```
p_prior_mutations(
  rates.matrix,
  mean.n.mut,
  ref_seq,
  save = FALSE,
  output_file = "tablePriorMutations.txt"
)
```

Arguments

| | |
|--------------|--|
| rates.matrix | Mutation rate matrix: 4x5 matrix, each row/col representing a nucleotide (col adds deletion), and the values is the mutational rate from row to col. |
| mean.n.mut | Mean number of mutations expected (one number). |
| ref_seq | DNAStrngSet containing the true reference sequence. |
| save | Logical. Should data be saved in a output_file? |
| output_file | File name for output, if save=TRUE. |

Value

Data frame with columns wt.base (wild type nucleotide), nucleotide (mutated nucleotide), p_mutation (probaility of mutation)

Examples

```

refseq_fasta <- system.file("extdata", "ref_seq.fasta", package = "single")
ref_seq <- Biostrings::subseq(Biostrings::readDNAStringSet(refseq_fasta), 1,10)
train_reads_example <- system.file("extdata", "train_seqs_500.sorted.bam",
                                   package = "single")
counts_pnq <- pileup_by_QUAL(train_reads_example, pos_start=1, pos_end=10)
p_prior_mutations <- p_prior_mutations(rates.matrix = mutation_rate,
                                       mean.n.mut = 5, ref_seq = ref_seq)
head(p_prior_mutations)

```

single_consensus_byBarcode

Compute SINGLE consensus

Description

Main function to compute consensus after correcting reads by a SINGLE model.

Usage

```

single_consensus_byBarcode(
  barcodes_table,
  sequences,
  readID_col = 1,
  bcID_col = 2,
  header = TRUE,
  dec = ".",
  sep = " ",
  verbose = TRUE
)

```

Arguments

barcodes_table data.frame or file name containing the names of the reads and the barcode associated (or any grouping tag).

sequences QualityScaledDNAStringSet or fastq file name. Contains sequences from which compute weighted consensus.

readID_col, bcID_col Numeric. Columns where the reads id and the barcode (or grouping tag) are, in the barcodes_table

header, dec, sep Arguments for read.table(barcodes_table)

verbose Logical.

Value

DNAStringSet with consensus sequences

Examples

```

refseq_fasta = system.file("extdata", "ref_seq.fasta", package = "single")
ref_seq <- Biostrings::subseq(Biostrings::readDNASTringSet(refseq_fasta), 1,10)
train_file <- system.file("extdata", "train_example.txt", package = "single")
train <- read.table(train_file, header=TRUE)
lib_example = system.file("extdata", "test_sequences.sorted.bam", package = "single")
corrected_reads <- single_evaluate(bamfile = lib_example,
                                single_fits = train, ref_seq = ref_seq,
                                pos_start=1, pos_end=10, gaps_weights = "minimum")
barcodes = system.file("extdata", "Barcodes_table.txt", package = "single")
consensus <- single_consensus_byBarcode(
  barcodes_table = barcodes,
  sequences = corrected_reads,
  verbose = FALSE)

```

| | |
|-----------------|------------------------------|
| single_evaluate | <i>Evaluate SINGLE model</i> |
|-----------------|------------------------------|

Description

Main function to evaluate a gene library using a SINGLE model.

Usage

```

single_evaluate(
  bamfile,
  single_fits,
  ref_seq,
  pos_start = NULL,
  pos_end = NULL,
  gaps_weights,
  save = FALSE,
  output_file,
  verbose = FALSE
)

```

Arguments

| | |
|-------------|--|
| bamfile | File containing the counts per position returned by samtools mpileup |
| single_fits | Results of the SINGLE model as returned by single_train(). It can be either the output data.frame or the saved file. |
| ref_seq | DNASTringSet containing the true reference sequence |
| pos_start | Numeric. Position to start analyzing, counting starts from 1 and it refers to reference used for minimap2 alignment. |
| pos_end | Numeric. Position to stop analyzing, counting starts from 1 and it refers to reference used for minimap2 alignment. |

| | |
|--------------|---|
| gaps_weights | One of "minimum","none","mean". How to assign qscores to deletions. |
| save | Logical. Should data be saved in a output_file? |
| output_file | File name for output, if save=TRUE. |
| verbose | Logical |

Details

Before running `single_evaluate_function` you have to align your INPUT data to a REFERENCE using `minimap2` and count the nucleotides per position using `samtools` using these lines:

```
minimap2 -ax map-ont --sam-hit-only REFERENCE.fasta INPUT.fastq >ALIGNMENT.sam
samtools view -S -b ALIGNMENT.sam > ALIGNMENT.bam
samtools sort ALIGNMENT.bam -o ALIGNMENT.sorted.bam
samtools mpileup -Q 0 ALIGNMENT.sorted.bam > COUNTS.txt
```

Value

Creates file `output_prefix_corrected.txt` with the Qscores re-scaled by SINGLE. Columns are SeqID position nucleotide isWT original_quality p_SINGLE

Examples

```
refseq_fasta <- system.file("extdata", "ref_seq.fasta", package = "single")
ref_seq <- Biostrings::subseq(Biostrings::readDNASTringSet(refseq_fasta), 1,10)
train_file <- system.file("extdata", "train_example.txt", package = "single")
train <- read.table(train_file, header=TRUE)
test_reads_example <- system.file("extdata", "test_sequences.sorted.bam",
  package = "single")
corrected_reads <- single_evaluate(bamfile = test_reads_example,
  single_fits = train,ref_seq = ref_seq,
  pos_start=1,pos_end=10,gaps_weights = "minimum")
corrected_reads
```

| | |
|--------------|---------------------------|
| single_train | <i>Train SINGLE model</i> |
|--------------|---------------------------|

Description

Main function to train a SINGLE model in a set of reads of a reference / wild type sequence. To get the input data you will need to run before a `minimap2` alignment and `samtools` counts.

Usage

```
single_train(
  bamfile,
  output = "results",
  refseq_fasta,
  rates.matrix = NULL,
  mean.n.mutations = NULL,
  pos_start = NULL,
  pos_end = NULL,
  verbose = TRUE,
  save_partial = FALSE,
  save_final = FALSE
)
```

Arguments

| | |
|------------------|--|
| bamfile | File containing the counts per position returned by samtools mpileup |
| output | String. Prefix for output files |
| refseq_fasta | Fasta file containing reference sequence |
| rates.matrix | Mutation rate matrix: 4x5 matrix, each row/col representing a nucleotide (col adds deletion), and the values is the mutational rate from row to col. |
| mean.n.mutations | Mean number of mutations expected (one number). |
| pos_start | Numeric. Position to start analyzing, counting starts from 1 and it refers to reference used for minimap2 alignment. |
| pos_end | Numeric. Position to stop analyzing, counting starts from 1 and it refers to reference used for minimap2 alignment. |
| verbose | Logical. |
| save_partial | Logical. Should partial results be saved in files? |
| save_final | Logical. Should final fits be saved in a file? |

Details

Before running `single_train_function` you have to align your INPUT data to a REFERENCE using `minimap2` and count the nucleotides per position using `samtools` using these lines:

```
minimap2 -ax map-ont --sam-hit-only REFERENCE.fasta INPUT.fastq >ALIGNMENT.sam
```

```
samtools view -S -b ALIGNMENT.sam > ALIGNMENT.bam
```

```
samtools sort ALIGNMENT.bam -o ALIGNMENT.sorted.bam
```

```
samtools mpileup -Q 0 ALIGNMENT.sorted.bam > COUNTS.txt
```

Value

Creates file `output_prefix_single_results.txt` with SINGLE training results.

Examples

```

refseq_fasta<- system.file("extdata", "ref_seq.fasta", package = "single")
train_reads_example <- system.file("extdata", "train_seqs_500.sorted.bam",
                                   package = "single")
train <- single_train(bamfile=train_reads_example,
                    refseq_fasta=refseq_fasta,
                    rates.matrix=mutation_rate,mean.n.mutations=5.4,
                    pos_start=1,pos_end=10)
print(head(train))

```

| | |
|--------------------|-----------------------------------|
| weighted_consensus | <i>Compute consensus sequence</i> |
|--------------------|-----------------------------------|

Description

This is an auxiliary function in single package. It computes consensus from a data.frame as the one returned by single_evaluate()

Usage

```
weighted_consensus(df, cutoff_prob = 0.2)
```

Arguments

| | |
|-------------|---|
| df | data.frame with the columns: nucleotide, probability, position |
| cutoff_prob | Numeric. Nucleotides with probability below this number will be removed from consensus computation. |

Value

Character vector, consensus sequence

Examples

```

fastq_seqs_example <- system.file("extdata", "test_sequences.fastq",package = "single")
seqs_example <- Biostrings::readQualityScaledDNASTringSet(fastq_seqs_example)
# Using single weights
data_barcode = data.frame(
  nucleotide = unlist(sapply(as.character(seqs_example),strsplit, split="")),
  p_SINGLE=unlist(1-as(Biostrings::quality(seqs_example),"NumericList")),
  pos=rep(1:Biostrings::width(seqs_example[1]),length(seqs_example)))
weighted_consensus(df = data_barcode, cutoff_prob = 0.9)
# Replacing weights by ones
data_barcode = data.frame(
  nucleotide = unlist(sapply(as.character(seqs_example),strsplit, split="")),
  p_SINGLE=1,pos=rep(1,sum(Biostrings::width(seqs_example))))
weighted_consensus(df = data_barcode, cutoff_prob = 0)

```

Index

- * **datasets**
 - bases, [2](#)
 - mutation_rate, [6](#)
- bases, [2](#)
- evaluate_fits, [2](#)
- fit_logregr, [3](#)
- glm.predict., [5](#)
- list_mismatches, [5](#)
- mutation_rate, [6](#)
- p_prior_errors, [7](#)
- p_prior_mutations, [8](#)
- pileup_by_QUAL, [6](#)
- single_consensus_byBarcode, [9](#)
- single_evaluate, [10](#)
- single_train, [11](#)
- weighted_consensus, [13](#)