

# Package: single (via r-universe)

July 2, 2024

**Type** Package

**Title** Accurate consensus sequence from nanopore reads of a gene library

**Version** 1.9.0

**biocViews** Software, Sequencing

**Depends** R (>= 4.0)

**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, tidyrr, utils

**Suggests** BiocStyle, knitr, rmarkdown

**VignetteBuilder** knitr

**RoxygenNote** 7.2.3

**Repository** <https://bioc.r-universe.dev>

**RemoteUrl** <https://github.com/bioc/single>

**RemoteRef** HEAD

**RemoteSha** e9ba58a6f5350dba3ed0621cbfc59147298687f4

## Contents

ascii_v . . . . .	2
bases . . . . .	2

evaluate_fits . . . . .	3
fit_logregr . . . . .	4
glm.predict. . . . .	5
list_mismatches . . . . .	6
mutation_rate . . . . .	6
pileup_by_QUAL . . . . .	7
p_prior_errors . . . . .	8
p_prior_mutations . . . . .	8
single_consensus_byBarcode . . . . .	9
single_evaluate . . . . .	10
single_train . . . . .	12
weighted_consensus . . . . .	13

<b>Index</b>	<b>15</b>
--------------	-----------

---

ascii_v	<i>ASCII code</i>
---------	-------------------

---

### Description

Vector ascii

### Usage

ascii\_v

### Format

Named character vector of length 94. Names are ascii character and value is the probability of error.

---

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

---

### Description

Vector A C G T -

### Usage

bases

### Format

Character vector of length 5



---

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,
  keep_fit_quality = FALSE
)
```

### Arguments

counts_pnq	Data frame with columns position nucleotide quality counts, as returned by pileup_by_QUAL
ref_seq	DNAStringSet containing the true reference sequence.
p_prior_errors	Data frame with columns position nucleotide prior.error, as the one returned by p_prior_errors().
p_prior_mutations	Data frame with columns wt.base, nucleotide and p_mutation (probability of mutation), as the one returned by p_prior_mutations().
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.
keep_fit_quality	Logical. Should parameters related to the quality of the fit be returned in extra columns of the output?

### 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	<i>Lists mismatches between two DNAstrings</i>
-----------------	--

---

**Description**

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

**Usage**

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

**Arguments**

ref	DNAString, reference sequence.
seq	DNAString, 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      *Computes prior probability of mutations*

---

### 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	DNASTringSet 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 (probability 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**

```
pos_start=1
pos_end = 100
barcodes_file = system.file("extdata", "Barcodes_table.txt", package = "single")
reads_single = system.file("extdata", "corrected_seqs.fastq", package = "single")
single_consensus_byBarcode(barcodes_file, reads_single, verbose = FALSE)
```

---

single\_evaluate

*Evaluate SINGLE model*

---

**Description**

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

**Usage**

```
single_evaluate(
  bamfile,
  single_fits,
  refseq_fasta,
  pos_start = NULL,
  pos_end = NULL,
  gaps_weights,
  save = FALSE,
  output_file,
  verbose = FALSE,
  save_original_scores = 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.
refseq_fasta	Fasta file containing 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
save_original_scores	Logical. Should original Qscores be saved? If TRUE, they are stored in file whose name finishes in _original.fastq

**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_10bases.fasta", package = "single")
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, refseq_fasta = refseq_fasta,
  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`      *Compute consensus sequence*

---

**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**

<code>df</code>	data.frame with the columns: nucleotide, probability, position
<code>cutoff_prob</code>	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**
  - ascii\_v, [2](#)
  - bases, [2](#)
  - mutation\_rate, [6](#)
- ascii\_v, [2](#)
- bases, [2](#)
- evaluate\_fits, [3](#)
- fit\_logregr, [4](#)
- glm.predict., [5](#)
- list\_mismatches, [6](#)
- mutation\_rate, [6](#)
- p\_prior\_errors, [8](#)
- p\_prior\_mutations, [8](#)
- pileup\_by\_QUAL, [7](#)
- single\_consensus\_byBarcode, [9](#)
- single\_evaluate, [10](#)
- single\_train, [12](#)
- weighted\_consensus, [13](#)