

Package: MIC (via r-universe)

March 9, 2025

Title Analysis of Antimicrobial Minimum Inhibitory Concentration Data

Version 1.0.2

Description Analyse, plot, and tabulate antimicrobial minimum inhibitory concentration (MIC) data. Validate the results of an MIC experiment by comparing observed MIC values to a gold standard assay, in line with standards from the International Organization for Standardization (2021) <https://www.iso.org/standard/79377.html>. Perform MIC prediction from whole genome sequence data stored in the Pathosystems Resource Integration Center (2013) [doi:10.1093/nar/gkt1099](https://doi.org/10.1093/nar/gkt1099) database or locally.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.2

Imports AMR, glue, readr, dplyr, Rcpp, data.table, Biostrings, stringr, rlang, tidyr, future.apply, progressr, lemon, ggplot2, forcats, purrr, tibble

Depends R (>= 4.1.0)

LazyData true

LinkingTo Rcpp

Suggests testthat (>= 3.0.0), xgboost, flextable, caret, lifecycle, future

Config/testthat/edition 3

URL <https://github.com/agerada/MIC>

BugReports <https://github.com/agerada/MIC/issues>

NeedsCompilation yes

Author Alessandro Gerada [aut, cre, cph]
(<https://orcid.org/0000-0002-6743-4271>)

Maintainer Alessandro Gerada <alessandro.gerada@liverpool.ac.uk>

Repository CRAN

Date/Publication 2025-02-07 09:40:02 UTC

Config/pak/sysreqs libicu-dev libssl-dev libx11-dev

Contents

bias	3
clean_raw_mic	3
combined_file_system	4
compare_mic	5
compare_sir	6
download_patric_db	7
ecoffs	8
essential_agreement	8
example_mics	9
fill_dilution_levels	10
force_mic	11
genomes_to_kmer_libsvm	12
genome_to_libsvm	13
get_mic	14
kmers	15
load_patric_db	17
mic_censor	17
mic_range	18
mic_r_breakpoint	19
mic_s_breakpoint	20
mic_uncensor	20
move_files	22
plot.mic_validation	23
print.mic_validation	24
print.mic_validation_summary	25
pull_PATRIC_genomes	25
qc_in_range	26
qc_on_target	27
replace_multiple_slashes	28
reverse_complement	29
split_and_combine_files	29
squeezed_index_to_str	31
squeezed_mers	32
standardise_mic	32
summary.mic_validation	34
table	34
tidy_patric_meta_data	35
train_test_filesystem	36
unsqueezed_index_to_str	38
unsqueezed_mers	38
xgb.cv.lowmem	39

Index

41

bias	<i>Calculate MIC bias</i>
------	---------------------------

Description

Calculate the bias between two AMR::mic vectors. The bias is calculated as the percentage of test MICs that are above the gold standard MICs minus the percentage of test MICs that are below the gold standard MICs.

Usage

```
bias(gold_standard, test)
```

Arguments

```
gold_standard  AMR::mic vector
test           AMR::mic vector
```

Value

numeric value

References

International Organization for Standardization. ISO 20776-2:2021 Available from: <https://www.iso.org/standard/79377.html>

Examples

```
gold_standard <- c("<0.25", "8", "64", ">64")
test <- c("<0.25", "2", "16", "64")
bias(gold_standard, test)
```

clean_raw_mic	<i>Clean up raw MIC for use as a feature</i>
---------------	--

Description

Removes leading "=" which can sometimes be present in raw MIC results. Also converts co-trimoxazole to trimethprim component only.

Usage

```
clean_raw_mic(mic)
```

Arguments

```
mic           character containing MIC/s
```

Value

character of clean MIC/s

Examples

```
clean_raw_mic(c("=>64", "0.25/8.0"))
```

combined_file_system *Combine train and test filesystem into single folder*

Description

This function reorganises files that have been split into train and test directories using `train_test_filesystem()` back into a single directory. This is a convenience function to reverse the effects of `train_test_filesystem()`.

Usage

```
combined_file_system(  
  path_to_folders,  
  file_ext,  
  train_folder = "train",  
  test_folder = "test",  
  overwrite = FALSE  
)
```

Arguments

<code>path_to_folders</code>	path containing test and train folders; files will be moved here
<code>file_ext</code>	file extension to filter
<code>train_folder</code>	train folder subdirectory name
<code>test_folder</code>	test folder subdirectory name
<code>overwrite</code>	force overwrite of files that already exist

Value

Logical vector, indicated success or failure for each file

Examples

```
set.seed(123)  
# create 10 random DNA files  
tmp_dir <- tempdir()  
# remove any existing .fna files  
file.remove(  
  list.files(tmp_dir, pattern = "*.fna", full.names = TRUE)  
)
```

```

for (i in 1:10) {
writeLines(paste0(">", i, "\n", paste0(sample(c("A", "T", "C", "G"),
  100, replace = TRUE), collapse = "")), file.path(tmp_dir, paste0(i, ".fna")))
}

# split files into train and test directories
paths <- train_test_filesystem(tmp_dir,
                              file_ext = "fna",
                              split = 0.8,
                              shuffle = TRUE,
                              overwrite = TRUE)

# combine files back into a single directory
combined_file_system(tmp_dir, "fna")
list.files(tmp_dir)

```

compare_mic

Compare and validate MIC values

Description

This function compares an vector of MIC values to another. Generally, this is in the context of a validation experiment – an investigational assay or method (the "test") is compared to a gold standard. The rules used by this function are in line with "ISO 20776-2:2021 Part 2: Evaluation of performance of antimicrobial susceptibility test devices against reference broth micro-dilution."

There are two levels of detail that are provided. If only the MIC values are provided, the function will look for essential agreement between the two sets of MIC. If the organism and antibiotic arguments are provided, the function will also calculate the categorical agreement using EUCAST breakpoints (or, if breakpoint not available and `accept_ecoff = TRUE`, ECOFFs).

The function returns a special dataframe of results, which is also an `mic_validation` object. This object can be summarised using `summary()` for summary metrics, plotted using `plot()` for an essential agreement confusion matrix, and tabulated using `table()`.

Usage

```

compare_mic(
  gold_standard,
  test,
  ab = NULL,
  mo = NULL,
  accept_ecoff = FALSE,
  simplify = TRUE
)

```

Arguments

`gold_standard` vector of MICs to compare against.

`test` vector of MICs that are under investigation

ab character vector (same length as MIC) of antibiotic names (optional)
 mo character vector (same length as MIC) of microorganism names (optional)
 accept_ecoff if TRUE, ECOFFs will be used when no clinical breakpoints are available
 simplify if TRUE, MIC values will be coerced into the closest halving dilution (e.g., 0.55 will be converted to 0.5)

Value

S3 mic_validation object

Examples

```

# Just using MIC values only
gold_standard <- c("<0.25", "8", "64", ">64")
test <- c("<0.25", "2", "16", "64")
val <- compare_mic(gold_standard, test)
summary(val)

# Using MIC values and antibiotic and organism names
gold_standard <- c("<0.25", "8", "64", ">64")
test <- c("<0.25", "2", "16", "64")
ab <- c("AMK", "AMK", "AMK", "AMK")
mo <- c("B_ESCHR_COLI", "B_ESCHR_COLI", "B_ESCHR_COLI", "B_ESCHR_COLI")
val <- compare_mic(gold_standard, test, ab, mo)
"error" %in% names(val) # val now has categorical agreement

```

compare_sir

Compare SIR results and generate categorical agreement

Description

Compare two AMR::sir vectors and generate a categorical agreement vector with the following levels: M (major error), vM (very major error), m (minor error). The error definitions are:

1. Major error (M): The test result is resistant (R) when the gold standard is susceptible (S).
2. vM (very major error): The test result is susceptible (S) when the gold standard is resistant (R).
3. Minor error (m): The test result is intermediate (I) when the gold standard is susceptible (S) or resistant (R), or vice versa.

Usage

```
compare_sir(gold_standard, test)
```

Arguments

gold_standard Susceptibility results in AMR::sir format
 test Susceptibility results in AMR::sir format

Value

factor vector with the following levels: M, vM, m.

Examples

```
gold_standard <- c("S", "R", "I", "I")
gold_standard <- AMR::as.sir(gold_standard)
test <- c("S", "I", "R", "R")
test <- AMR::as.sir(test)
compare_sir(gold_standard, test)
```

download_patric_db *Download PATRIC database*

Description

Download PATRIC database

Usage

```
download_patric_db(save_path, ftp_path = patric_ftp_path, overwrite = FALSE)
```

Arguments

- save_path Save path (should be .txt)
- ftp_path PATRIC database FTP path to download
- overwrite Force overwrite

Value

TRUE if successful, FALSE if failure.

Examples

```
download_patric_db(tempfile())
```

 ecoffs

ECOFF data

Description

A dataset containing the epidemiological cut-off values (ECOFFs) for different antibiotics and microorganisms. Currently, only the ECOFF values for *Escherichia coli* are included.

Usage

ecoffs

Format

ecoffs:

A data frame with 85 rows and 25 columns:

organism Microorganism code in AMR::mo format

antibiotic Antibiotic code in AMR::ab format

0.002:512 Counts of isolates in each concentration "bin"

Distributions see EUCAST documentation below

Observations Number of observations

(T)ECOFF see EUCAST documentation below

Confidence interval see EUCAST documentation below

Source

EUCAST https://www.eucast.org/mic_and_zone_distributions_and_ecoffs

These data have (or this document, presentation or video has) been produced in part under ECDC service contracts and made available by EUCAST at no cost to the user and can be accessed on the EUCAST website www.eucast.org. The views and opinions expressed are those of EUCAST at a given point in time. EUCAST recommendations are frequently updated and the latest versions are available at www.eucast.org.

 essential_agreement

Essential agreement for MIC validation

Description

Essential agreement calculation for comparing two MIC vectors.

Usage

essential_agreement(x, y, coerce_mic = TRUE, mode = "categorical")

Arguments

x	AMR::mic or coercible
y	AMR::mic or coercible
coerce_mic	convert to AMR::mic
mode	Categorical or numeric

Details

Essential agreement is a central concept in the comparison of two sets of MIC values. It is most often used when validating a new method against a gold standard. This function reliably performs essential agreement in line with ISO 20776-2:2021. The function can be used in two modes: categorical and numeric. In categorical mode, the function will use traditional MIC concentrations to determine the MIC (therefore it will use `force_mic()` to convert both x and y to a clean MIC – see `?force_mic()`). In numeric mode, the function will compare the ratio of the two MICs. In most cases, categorical mode provides more reliable results. Values within +/- 2 dilutions are considered to be in essential agreement.

Value

logical vector

References

International Organization for Standardization. ISO 20776-2:2021 Available from: <https://www.iso.org/standard/79377.html>

Examples

```
x <- AMR::as.mic(c("<0.25", "8", "64", ">64"))
y <- AMR::as.mic(c("<0.25", "2", "16", "64"))
essential_agreement(x, y)
# TRUE FALSE FALSE TRUE
```

example_mics

Example MIC data

Description

Example minimum inhibitory concentration validation data for three antimicrobials on Escherichia coli strains. This data is synthetic and generated to give an example of different MIC distribution.

Usage

example_mics

Format

example_mics:

A data frame with 300 rows and 4 columns:

gs Gold standard MICs

test Test MICs

mo Microorganism code in AMR::mo format

ab Antibiotic code in AMR::ab format

Source

Synthetic data

fill_dilution_levels *Fill MIC dilution levels*

Description

Fill MIC dilution levels

Usage

```
fill_dilution_levels(x, cap_upper = TRUE, cap_lower = TRUE, as.mic = TRUE)
```

Arguments

x	MIC vector
cap_upper	If True, will the top level will be the highest MIC dilution in x
cap_lower	If True, will the bottom level will be the lowest MIC dilution in x
as.mic	By default, returns an ordered factor. Set as.mic = TRUE to return as AMR::mic

Value

ordered factor (or AMR::mic if as.mic = TRUE)

Examples

```
# use in combination with droplevels to clean up levels:  
x <- AMR::as.mic(c("<0.25", "8", "64", ">64"))  
x <- droplevels(x)  
fill_dilution_levels(x)
```

`force_mic`*Force MIC-like into MIC-compatible format*

Description

Convert a value that is "almost" an MIC into a valid MIC value.

Usage

```
force_mic(  
  value,  
  levels_from_AMR = FALSE,  
  max_conc = 512,  
  min_conc = 0.002,  
  method = "closest",  
  prefer = "max"  
)
```

Arguments

<code>value</code>	vector of MIC-like values (numeric or character)
<code>levels_from_AMR</code>	conform to <code>AMR::as.mic</code> levels
<code>max_conc</code>	maximum concentration to force to
<code>min_conc</code>	minimum concentration to force to
<code>method</code>	method to use when forcing MICs (closest or round_up)
<code>prefer</code>	where value is in between MIC (e.g., 24mg/L) chose the higher MIC ("max") or lower MIC ("min"); only applies to method = "closest"

Details

Some experimental or analytical conditions measure MIC (or surrogate) in a way that does not fully conform to traditional MIC levels (i.e., concentrations). This function allows these values to be coerced into an MIC value that is compatible with the `AMR::mic` class. When using method = "closest", the function will choose the closest MIC value to the input value (e.g., 2.45 will be coerced to 2). When using method = "round up", the function will round up to the next highest MIC value (e.g., 2.45 will be coerced to 4). "Round up" is technically the correct approach if the input value was generated from an experiment that censored between concentrations (e.g., broth or agar dilution). However, "closest" may be more appropriate in some cases.

Value

`AMR::as.mic` compatible character

Examples

```
force_mic(c("2.32", "<4.12", ">1.01"))
```

`genomes_to_kmer_libsvm`*Convert genomes to kmers in libsvm format*

Description

Raw genome data (pre- or post-assembly) is usually transformed by k-mer counting prior to machine learning (ML). XGBoost is a popular ML algorithm for this problem, due to its scalability to high dimensional data. This function converts genomes to k-mer counts stored in XGBoost's preferred format, libsvm. Further information on the libsvm format is available at https://xgboost.readthedocs.io/en/stable/tutorials/input_format.html. Briefly, libsvm is effectively a text file that stores data points as x:y pairs, where x is the feature index, and y is the feature value. Each observation is stored on its own line, with the first column reserved for labels. Labels can be provided later, during data import.

This function converts each individual genome to an individual libsvm text file of k-mer counts (therefore, each .txt file will be 1 line long). This function supports parallel processing using the by setting an appropriate `future::plan()` (usually `future::multisession`) — each genome is processed in parallel. To monitor progress, use the `progressr` package by wrapping the function in `with_progress`.

Although XGBoost can load a multiple .txt (libsvm) files by providing the directory as an input, this is generally not recommended as order of import cannot be guaranteed and probably depends on filesystem. Instead, it is recommended that this function is combined with `split_and_combine_files()` which generates a single .txt file (with the order of observations guaranteed and stored in a .csv file).

Usage

```
genomes_to_kmer_libsvm(  
  source_dir,  
  target_dir,  
  k = 3,  
  canonical = TRUE,  
  squeeze = FALSE,  
  ext = ".fna"  
)
```

Arguments

<code>source_dir</code>	directory containing genomes
<code>target_dir</code>	target directory to store kmers in libsvm format
<code>k</code>	k-mer length
<code>canonical</code>	only count canonical kmers
<code>squeeze</code>	remove non-canonical kmers
<code>ext</code>	file extension to filter

Value

TRUE if successful

See Also

to convert a single genome, use [genome_to_libsvm\(\)](#)

Examples

```
set.seed(123)
# create 10 random DNA files
tmp_dir <- tempdir()
# remove any existing .fna files
file.remove(
  list.files(tmp_dir, pattern = "*.fna", full.names = TRUE)
)
for (i in 1:10) {
  writeLines(paste0(">", i, "\n", paste0(sample(c("A", "T", "C", "G"),
    100, replace = TRUE), collapse = "")), file.path(tmp_dir, paste0(i, ".fna")))
}

tmp_target_dir <- file.path(tmp_dir, "kmers")
unlink(tmp_target_dir, recursive = TRUE)

# convert genomes to k-mers
future::plan(future::sequential) # use multisection for parallel processing
progressr::with_progress(
  genomes_to_kmer_libsvm(tmp_dir, tmp_target_dir, k = 3)
)

# check the output
list.files(tmp_target_dir)
readLines(list.files(tmp_target_dir, full.names = TRUE)[1])
```

genome_to_libsvm

Converts a genome to kmers stored in libsvm format on disk

Description

This function converts a single genome to a libsvm file containing kmer counts. The libsvm format will be as follows:

```
label 1:count 2:count 3:count ...
```

Label is optional and defaults to 0. The kmer counts are indexed by the kmer index, which is the lexicographically sorted index of the kmer. Libsvm is a sparse format.

Usage

```
genome_to_libsvm(
  x,
  target_path,
  label = as.character(c("0")),
  k = 3L,
  canonical = TRUE,
  squeeze = FALSE
)
```

Arguments

x	genome in string format
target_path	path to store libsvm file (.txt)
label	libsvm label
k	kmer length
canonical	only record canonical kmers (i.e., the lexicographically smaller of a kmer and its reverse complement)
squeeze	remove non-canonical kmers

Value

boolean indicating success

See Also

For multiple genomes in a directory, processed in parallel, see [genomes_to_kmer_libsvm\(\)](#)

For more details on libsvm format, see https://xgboost.readthedocs.io/en/stable/tutorials/input_format.html

Examples

```
temp_libsvm_path <- tempfile(fileext = ".txt")
genome_to_libsvm("ATCGCAGT", temp_libsvm_path)
readLines(temp_libsvm_path)
```

get_mic

Get MIC meta-data from feature database

Description

This function helps extract MICs from a database of results. It is compatible with the PATRIC meta data format when used on a tidy_patric_db object, created using tidy_patric_db().

If more than one MIC is present for a particular observation, the function can return the higher MIC by setting prefer_high_mic = TRUE. If prefer_high_mic = FALSE, the lower MIC will be returned.

Usage

```
get_mic(
  x,
  ids,
  ab_col,
  id_col = NULL,
  as_mic = TRUE,
  prefer_high_mic = TRUE,
  simplify = TRUE
)
```

Arguments

x	dataframe containing meta-data
ids	vector of IDs to get meta-data for
ab_col	column name containing MIC results
id_col	column name containing IDs
as_mic	return as AMR::as.mic
prefer_high_mic	where multiple MIC results per ID, prefer the higher MIC
simplify	return as vector of MICs (vs dataframe)

Value

vector containing MICs, or dataframe of IDs and MICs

Examples

```
df <- data.frame(genome_id = c("a_12", "b_42", "x_21", "x_21", "r_75"),
  gentamicin = c(0.25, 0.125, 32.0, 16.0, "<0.0125"))
get_mic(df,
  ids = c("b_42", "x_21"),
  ab_col = "gentamicin",
  id_col = "genome_id",
  as_mic = FALSE,
  prefer_high_mic = TRUE,
  simplify = TRUE)
```

 kmers

Generates genome kmers

Description

Generates genome kmers

Usage

```

kmers(
  x,
  k = 3L,
  simplify = FALSE,
  canonical = TRUE,
  squeeze = FALSE,
  anchor = TRUE,
  clean_up = TRUE,
  key_as_int = FALSE,
  starting_index = 1L
)

```

Arguments

x	genome in string format
k	kmer length
simplify	returns a numeric vector of kmer counts, without associated string. This is useful to save memory, but should always be used with anchor = true.
canonical	only record canonical kmers (i.e., the lexicographically smaller of a kmer and its reverse complement)
squeeze	remove non-canonical kmers
anchor	includes unobserved kmers (with counts of 0). This is useful when generating a dense matrix where kmers of different genomes align.
clean_up	only include valid bases (ACTG) in kmer counts (excludes non-coding results such as N)
key_as_int	return kmer index (as "kmer_index") rather than the full kmer string. Useful for index-coded data structures such as libsvm.
starting_index	the starting index, only used if key_as_int = TRUE.

Value

list of kmer values, either as a list of a single vector (if simplify = TRUE), or as a named list containing "kmer_string" and "kmer_value".

Examples

```
kmers("ATCGCAGT")
```

load_patric_db	<i>Load PATRIC database</i>
----------------	-----------------------------

Description

Load PATRIC database

Usage

```
load_patric_db(x = patric_ftp_path)
```

Arguments

x Character path to local or ftp path (.txt or .rds), or data.frame object.

Value

PATRIC database (S3 class 'patric_db')

Examples

```
patric_db <- load_patric_db() # will get from PATRIC ftp
```

```
# make data.frame with single row
p <- data.frame(genome_id = 1,
                genome_name = "E. coli",
                antibiotic = "amoxicillin",
                measurement = 2.0,
                measurement_unit = "mg/L",
                laboratory_typing_method = "Agar dilution",
                resistant_phenotype = "R")
load_patric_db(p)
```

mic_censor	<i>Censor MIC values</i>
------------	--------------------------

Description

MIC datasets often arise from different laboratories or experimental conditions. In practice, this means that there can be different levels of censoring (\leq and $>$) within the data. This function can be used to harmonise the dataset to a single level of censoring. The function requires a set of rules that specify the censoring levels (see example).

Usage

```
mic_censor(mic, ab, mo, rules)
```

Arguments

mic	MIC (coercible to AMR::as.mic)
ab	antibiotic name (coercible to AMR::as.ab)
mo	microorganism name (coercible to AMR::as.mo)
rules	sensor rules - named list of pathogen (in AMR::as.mo code) to antibiotic (in AMR::as.ab code) to censoring rules. The censoring rules should provide a min or max value to censor MICs to. See example for more.

Value

censored MIC values (S3 mic class)

Examples

```
example_rules <- list("B_ESCHR_COLI" = list(
  "AMK" = list(min = 2, max = 32),
  "CHL" = list(min = 4, max = 64),
  "GEN" = list(min = 1, max = 16),
  "CIP" = list(min = 0.015, max = 4),
  "MEM" = list(min = 0.016, max = 16),
  "AMX" = list(min = 2, max = 64),
  "AMC" = list(min = 2, max = 64),
  "FEP" = list(min = 0.5, max = 64),
  "CAZ" = list(min = 1, max = 128),
  "TGC" = list(min = 0.25, max = 1)
))

mic_censor(AMR::as.mic(512),
  "AMK",
  "B_ESCHR_COLI",
  example_rules) == AMR::as.mic(">32")
```

mic_range

Generate dilution series

Description

Generate dilution series

Usage

```
mic_range(start = 512, dilutions = Inf, min = 0.002, precise = FALSE)
```

Arguments

start	starting (highest) concentration
dilutions	number of dilutions
min	minimum (lowest) concentration
precise	force range to be high precision (not usually desired behaviour)

Value

Vector of numeric concentrations

Examples

```
mic_range(128)
mic_range(128, dilutions = 21) # same results
```

mic_r_breakpoint	<i>R breakpoint for MIC</i>
------------------	-----------------------------

Description

R breakpoint for MIC

Usage

```
mic_r_breakpoint(mo, ab, accept_ecoff = FALSE, ...)
```

Arguments

mo	mo name (coerced using AMR::as.mo)
ab	ab name (coerced using AMR::as.ab)
accept_ecoff	if TRUE, ECOFFs will be used when no clinical breakpoints are available
...	additional arguments to pass to AMR::as.sir, which is used to calculate the R breakpoint

Value

MIC value

Examples

```
mic_r_breakpoint("B_ESCHR_COLI", "AMK")
mic_r_breakpoint("B_ESCHR_COLI", "CHL", accept_ecoff = TRUE)
```

mic_s_breakpoint	<i>S breakpoint for MIC</i>
------------------	-----------------------------

Description

S breakpoint for MIC

Usage

```
mic_s_breakpoint(mo, ab, accept_ecoff = FALSE, ...)
```

Arguments

mo	mo name (coerced using AMR::as.mo)
ab	ab name (coerced using AMR::as.ab)
accept_ecoff	if TRUE, ECOFFs will be used when no clinical breakpoints are available
...	additional arguments to pass to AMR::as.sir, which is used to calculate the S breakpoint

Value

MIC value

Examples

```
mic_s_breakpoint("B_ESCHR_COLI", "AMK")
mic_s_breakpoint("B_ESCHR_COLI", "CHL", accept_ecoff = TRUE)
```

mic_uncensor	<i>Uncensor MICs</i>
--------------	----------------------

Description

Uncensor MICs

Usage

```
mic_uncensor(
  mic,
  method = "scale",
  scale = 2,
  ab = NULL,
  mo = NULL,
  distros = NULL
)
```

Arguments

mic	vector of MICs to uncensor; will be coerced to MIC using <code>AMR::as.mic</code>
method	method to uncensor MICs (scale, simple, or bootstrap)
scale	scalar to multiply or divide MIC by (for method = scale)
ab	antibiotic name (for method = bootstrap)
mo	microorganism name (for method = bootstrap)
distros	dataframe of epidemiological distributions (only used, optionally, for method = bootstrap)

Details

Censored MIC data is generally unsuitable for modelling without some conversion of censored data. The default behaviour (method = scale) is to halve MICs under the limit of detection (\leq) and double MICs above the limit of detection ($>$). When used with method = simple, this function effectively just removes the censoring symbols, e.g., ≤ 2 becomes 2, and > 64 becomes 64.

The bootstrap method is the more complex of the three available methods. It attempts to use a second (uncensored) MIC distribution to sample values in the censored range. These values are then used to populate and uncensor the MIC data provided as input (mic). The second (uncensored) MIC distribution is ideally provided from similar experimental conditions. Alternatively, epidemiological distributions can be used. These distributions should be provided as a dataframe to the distros argument. The format for this dataframe is inspired by the EUCAST epidemiological distributions, see: https://www.eucast.org/mic_and_zone_distributions_and_ecoffs. The dataframe should contain columns for antimicrobial (converted using `AMR::as.ab`), organism (converted using `AMR::as.mo`), and MIC concentrations. An example is provided in the 'ecoffs' dataset available with this package. Currently, only Escherichia coli is available in this dataset. Each observation (row) consists of the frequency a particular MIC concentration is observed in the distribution. If such a dataframe is not provided to distros, the function will attempt to use 'ecoffs', but remains limited to E. coli.

Value

vector of MICs in `AMR::mic` format

References

https://www.eucast.org/mic_and_zone_distributions_and_ecoffs

Examples

```
mic_uncensor(c(">64.0", "<0.25", "8.0"), method = "scale", scale = 2)
```

move_files	<i>Move or copy files using logical vector</i>
------------	--

Description

This is simply a wrapper around `file.copy/file.rename` that allows for filtering by a logical vector (`move_which`). This can replicate the behaviour of a predicate function (see example), and may be easier to read.

Usage

```
move_files(source_dir, target_dir, move_which, ext = ".txt", copy = FALSE)
```

Arguments

<code>source_dir</code>	move from directory
<code>target_dir</code>	move to directory
<code>move_which</code>	logical vector to filter (or use TRUE to move all)
<code>ext</code>	file extension to filter
<code>copy</code>	copy files (rather than move)

Value

Logical vector, indicating success or failure for each file

Examples

```
set.seed(123)
# create 10 random DNA files
tmp_dir <- tempdir()
# remove any existing .fna files
file.remove(
  list.files(tmp_dir, pattern = "*.fna", full.names = TRUE)
)
for (i in 1:10) {
  writeLines(paste0(">", i, "\n", paste0(sample(c("A", "T", "C", "G"),
    100, replace = TRUE), collapse = "")), file.path(tmp_dir, paste0(i, ".fna")))
}

# move files with even numbers to a new directory
new_dir <- file.path(tempdir(), "even_files")
unlink(new_dir, recursive = TRUE)
move_files(tmp_dir,
  new_dir,
  move_which = as.integer(
    tools::file_path_sans_ext(
      list.files(tmp_dir, pattern = "*.fna"))) %% 2 == 0,
  ext = "fna")
list.files(new_dir)
```

plot.mic_validation *Plot MIC validation results*

Description

Plot MIC validation results

Usage

```
## S3 method for class 'mic_validation'
plot(
  x,
  match_axes = TRUE,
  add_missing_dilutions = TRUE,
  facet_wrap_ncol = NULL,
  facet_wrap_nrow = NULL,
  ...
)
```

Arguments

x	object generated using compare_mic
match_axes	Same x and y axis
add_missing_dilutions	Axes will include dilutions that are not
facet_wrap_ncol	Facet wrap into n columns by antimicrobial (optional, only available when more than one antimicrobial in validation)
facet_wrap_nrow	Facet wrap into n rows by antimicrobial (optional, only available when more than one antimicrobial in validation) represented in the data, based on a series of dilutions generated using mic_range().
...	additional arguments

Value

ggplot object

Examples

```
gold_standard <- c("<0.25", "8", "64", ">64")
test <- c("<0.25", "2", "16", "64")
val <- compare_mic(gold_standard, test)
plot(val)

# works with validation that includes categorical agreement
# categorical agreement is ignored
```

```
ab <- c("AMK", "AMK", "AMK", "AMK")
mo <- c("B_ESCHR_COLI", "B_ESCHR_COLI", "B_ESCHR_COLI", "B_ESCHR_COLI")
val <- compare_mic(gold_standard, test, ab, mo)
plot(val)

# if the validation contains multiple antibiotics, i.e.,
ab <- c("CIP", "CIP", "AMK", "AMK")
val <- compare_mic(gold_standard, test, ab, mo)
# the following will plot all antibiotics in a single plot (pooled results)
plot(val)
# use the faceting arguments to split the plot by antibiotic
plot(val, facet_wrap(ncol = 2))
```

print.mic_validation *Print MIC validation object*

Description

Print MIC validation object

Usage

```
## S3 method for class 'mic_validation'
print(x, ...)
```

Arguments

x	mic_validation object
...	additional arguments

Value

character

Examples

```
gold_standard <- c("<0.25", "8", "64", ">64")
test <- c("<0.25", "2", "16", "64")
val <- compare_mic(gold_standard, test)
print(val)
```

```
print.mic_validation_summary
      Print MIC validation summary
```

Description

Print MIC validation summary

Usage

```
## S3 method for class 'mic_validation_summary'
print(x, ...)
```

Arguments

x	mic_validation_summary object
...	additional arguments

Value

character

Examples

```
gold_standard <- c("<0.25", "8", "64", ">64")
test <- c("<0.25", "2", "16", "64")
val <- compare_mic(gold_standard, test)
print(summary(val))
```

```
pull_PATRIC_genomes  Automated download of genomes from PATRIC database
```

Description

Automated download of genomes from PATRIC database

Usage

```
pull_PATRIC_genomes(
  output_directory,
  taxonomic_name = NULL,
  database = patric_ftp_path,
  filter = "MIC",
  n_genomes = 0
)
```

Arguments

output_directory local directory to save to

taxonomic_name character of taxonomic bacterial name to download

database local or ftp path to PATRIC database, or loaded database using load_patric_db()

filter "MIC" or "disk" or "all" phenotypes

n_genomes number of genomes (0 = all)

Value

The number of failed downloads (i.e., 0 if all attempted downloads were successful).

Examples

```
pull_PATRIC_genomes(tempdir(),
  taxonomic_name = "Escherichia coli",
  filter = "MIC",
  n_genomes = 10)
```

qc_in_range

Check that MIC is within QC range

Description

Check whether MIC values are within acceptable range for quality control (QC). Every MIC experiment should include a control strain with a known MIC. The results of the experiment are only valid if the control strain MIC falls within the acceptable range. This function checks whether an MIC result is within the acceptable range given: 1) a control strain (usually identified as an ATCC or NCTC number), 2) an antibiotic name, and 3) a guideline (EUCAST or CLSI). The acceptable range is defined by 'QC_table', which is a dataset which is loaded with this package.

The source of the QC values is the WHONET QC Ranges and Targets available from the 'Antimicrobial Resistance Test Interpretation Engine' (AMRIE) repository: <https://github.com/AClark-WHONET/AMRIE>

Usage

```
qc_in_range(
  measurement,
  strain,
  ab,
  ignore_na = TRUE,
  guideline = "EUCAST",
  year = "2023"
)
```

Arguments

measurement	measured QC MIC
strain	control strain identifier (usually ATCC)
ab	antibiotic name (will be coerced to AMR::as.ab)
ignore_na	ignores NA (returns TRUE)
guideline	Guideline to use (EUCAST or CLSI)
year	Guideline year (version)

Value

logical vector

References

O'Brien TF, Stelling JM. WHONET: An Information System for Monitoring Antimicrobial Resistance. *Emerg Infect Dis.* 1995 Jun;1(2):66–66.

Examples

```
qc_in_range(AMR::as.mic(0.5), 25922, "GEN") == TRUE
qc_in_range(AMR::as.mic(8.0), 25922, "GEN") == FALSE
```

qc_on_target	<i>Check that QC measurement is at the required target</i> [Experimental]
--------------	--

Description

MIC experiments should include a control strain with a known MIC. The MIC result for the control strain should be a particular target MIC. This function checks whether the target MIC was achieved given: 1) a control strain (usually identified as an ATCC or NCTC number), 2) an antibiotic name, and 3) a guideline (EUCAST or CLSI).

Since QC target values are currently not publicly available in an easy to use format, this function takes a pragmatic approach – for most antibiotics and QC strains, the target is assumed to be the midpoint of the acceptable range. This approximation is not necessarily equal to the QC target reported by guideline setting bodies such as EUCAST. Therefore, this function is considered experimental and should be used with caution.

This function can be used alongside `qc_in_range()`, which checks whether the MIC is within the acceptable range.

The source of the QC values is the WHONET QC Ranges and Targets available from the 'Antimicrobial Resistance Test Interpretation Engine' (AMRIE) repository: <https://github.com/AClark-WHONET/AMRIE>

Usage

```
qc_on_target(
  measurement,
  strain,
  ab,
  ignore_na = TRUE,
  guideline = "EUCAST",
  year = "2023"
)
```

Arguments

measurement	measured QC MIC
strain	control strain identifier (usually ATCC)
ab	antibiotic name (will be coerced to AMR::as.ab)
ignore_na	ignores NA (returns TRUE)
guideline	Guideline to use (EUCAST or CLSI)
year	Guideline year (version)

Value

logical vector

References

O'Brien TF, Stelling JM. WHONET: An Information System for Monitoring Antimicrobial Resistance. *Emerg Infect Dis.* 1995 Jun;1(2):66–66.

Examples

```
qc_on_target(AMR::as.mic(0.5), 25922, "GEN") == TRUE
```

replace_multiple_slashes

Removes multiple slashes in a path or url

Description

Removes multiple slashes in a path or url

Usage

```
replace_multiple_slashes(path)
```

Arguments

path	character vector
------	------------------

Value

character vector of paths without duplicate slashes

reverse_complement *Reverse complement of DNA string*

Description

Reverse complement of DNA string

Usage

```
reverse_complement(dna)
```

Arguments

dna DNA string

Value

reverse complement of DNA string

Examples

```
reverse_complement("ATCG")
```

split_and_combine_files

Create test train files from a number of files

Description

This function combines files into a train and test set, stored on disk. It can be used in combination with `genomes_to_kmer_libsvm()` to create a dataset that can be loaded into XGBoost (either by first creating an `xgboost::DMatrix`, or by using the data argument in `xgboost::xgb.train()` or `xgboost::xgb.cv()`). The following three files will be created:

1. train.txt - the training data
2. test.txt - the testing data (if `split < 1`)
3. names.csv - a csv file containing the original filenames and their corresponding type (train or test)

The function will check if the data is already in the appropriate format and will not overwrite unless forced using the overwrite argument.

By providing 1.0 to the split argument, the function can be used to combine files without a train-test split. In this case, all the files will be classed as 'train', and there will be no 'test' data. This is useful if one wants to perform cross-validation using `xgboost::xgb.cv()` or `MIC::xgb.cv.lowmem()`. It is also possible to combine all data into train and then perform splitting after loading into an `xgboost::DMatrix`, using `xgboost::slice()`.

Usage

```
split_and_combine_files(
  path_to_files,
  file_ext = ".txt",
  split = 0.8,
  train_target_path = NULL,
  test_target_path = NULL,
  names_backup = NULL,
  shuffle = TRUE,
  overwrite = FALSE
)
```

Arguments

<code>path_to_files</code>	path containing files or vector of filepaths
<code>file_ext</code>	file extension to filter
<code>split</code>	train-test split
<code>train_target_path</code>	name of train file to save as (by default, will be train.txt in the path_to_files directory)
<code>test_target_path</code>	name of test file to save as (by default, will be test.txt in the path_to_files directory)
<code>names_backup</code>	name of file to save backup of filename metadata (by default, will be names.csv in the path_to_files directory)
<code>shuffle</code>	randomise prior to splitting
<code>overwrite</code>	overwrite target files

Value

named list of paths to created train/test files, original filenames

Examples

```
set.seed(123)
# create 10 random libsvm files
tmp_dir <- tempdir()
# remove any existing .txt files
```

```

file.remove(
list.files(tmp_dir, pattern = "*.txt", full.names = TRUE)
)
for (i in 1:10) {
  # each line is K: V
  writeLines(paste0(i, ": ", paste0(sample(1:100, 10, replace = TRUE),
collapse = " ")), file.path(tmp_dir, paste0(i, ".txt")))
}

# split files into train and test directories
paths <- split_and_combine_files(
  tmp_dir,
  file_ext = "txt",
  split = 0.8,
  train_target_path = file.path(tmp_dir, "train.txt"),
  test_target_path = file.path(tmp_dir, "test.txt"),
  names_backup = file.path(tmp_dir, "names.csv"),
  overwrite = TRUE)

readLines(paths[["train"]])

```

squeezed_index_to_str *Get str conversion of squeezed kmer using index*

Description

Get str conversion of squeezed kmer using index

Usage

```
squeezed_index_to_str(x, k, starting_index = 1L)
```

Arguments

x integer vector of kmer indices
k kmer length
starting_index starting index (libsvm is usually indexed starting at 1)

Value

vector of squeezed kmer strings

Examples

```
squeezed_index_to_str(2, k = 3)
```

squeezed_mers *Generates all permutations of squeezed kmers*

Description

Generates all permutations of squeezed kmers

Usage

```
squeezed_mers(k = 3L)
```

Arguments

k kmer length

Value

vector of squeezed kmers

Examples

```
squeezed_mers(3)
```

standardise_mic *Standardise MIC to control strain [Experimental]*

Description

MIC experiments are generally quality-controlled by including a control strain with a known MIC. The MIC result for the control strain should be a particular target MIC, or at least within an acceptable range. This function standardises a measured MIC to the target MIC given: 1) a control strain (usually identified as an ATCC or NCTC number), 2) an antibiotic name, and 3) a guideline (EUCAST or CLSI). The definition of standardisation in this context is to adjust the measured MIC based on the QC MIC. This is based on the following principles and assumption:

1. A measured MIC is composed of two components: the true MIC and a measurement error. The measurement error is considered to be inevitable when measuring MICs, and is likely to be further composed of variability in laboratory conditions and operator interpretation.
2. It is assumed that the MIC of the control strain in the experiment has also been affected by this error.

The standardisation applied by this function uses the measured QC strain MIC as a reference point, and scales the rest of the MICs to this reference. In general, this means that the MICs are doubled or halved, depending on the result of the QC MIC. A worked example is provided below and illustrates the transformation that this function applies.

There is no current evidence base for this approach, therefore, this function is considered experimental and should be used with caution.

Usage

```
standardise_mic(
  test_measurement,
  qc_measurement,
  strain,
  ab,
  prefer_upper = FALSE,
  ignore_na = TRUE,
  guideline = "EUCAST",
  year = "2023",
  force = TRUE
)
```

Arguments

test_measurement	Measured MIC to standardise
qc_measurement	Measured QC MIC to standardise to
strain	control strain identifier (usually ATCC)
ab	antibiotic name (will be coerced to AMR::as.ab)
prefer_upper	Where the target MIC is a range, prefer the upper value in the range
ignore_na	Ignore NA (returns AMR::NA_mic_)
guideline	Guideline to use (EUCAST or CLSI)
year	Guideline year (version)
force	Force into MIC-compatible format after standardisation

Value

AMR::mic vector

Examples

```
# Ref strain QC MIC for GEN is 0.5
standardise_mic(
  test_measurement = c(AMR::as.mic(">8.0"), # QC = 1, censored MIC remains censored
                      AMR::as.mic(4.0), # QC = 0.5 which is on target, so stays same
                      AMR::as.mic(2), # QC = 1, so scaled down to 1
                      AMR::as.mic(2)), # QC = 0.25, so scaled up to 8
  qc_measurement = c(AMR::as.mic(1),
                    AMR::as.mic(0.5),
                    AMR::as.mic(1),
                    AMR::as.mic(0.25)),
  strain = 25922,
  ab = AMR::as.ab("GEN"))
```

```
summary.mic_validation
```

Summary of MIC validation results

Description

Summarise the results of an MIC validation generated using `compare_mic()`.

Usage

```
## S3 method for class 'mic_validation'
summary(object, ...)
```

Arguments

```
object      S3 mic_validation object
...         further optional parameters
```

Value

S3 `mic_validation_summary` object

Examples

```
gold_standard <- c("<0.25", "8", "64", ">64")
test <- c("<0.25", "2", "16", "64")
val <- compare_mic(gold_standard, test)
summary(val)
# or, for more detailed results
as.data.frame(summary(val))
```

```
table
```

Table

Description

Table

Usage

```
table(x, ...)

## Default S3 method:
table(x, ...)

## S3 method for class 'mic_validation'
```

```
table(  
  x,  
  format = "flextable",  
  fill_dilutions = TRUE,  
  bold = TRUE,  
  ea_color = NULL,  
  gold_standard_name = "Gold Standard",  
  test_name = "Test",  
  ...  
)
```

Arguments

x	mic_validation S3 object
...	further arguments
format	simple or flextable
fill_dilutions	Fill dilutions that are not present in the data in order to match the y- and x- axes
bold	Bold cells where essential agreement is TRUE
ea_color	Background color for essential agreement cells
gold_standard_name	Name of the gold standard to display in output
test_name	Name of the test to display in output

Value

table or flextable object

Examples

```
gold_standard <- c("<0.25", "8", "64", ">64")  
test <- c("<0.25", "2", "16", "64")  
val <- compare_mic(gold_standard, test)  
table(val)
```

tidy_patric_meta_data *Tidy PATRIC data*

Description

Tidy PATRIC data

Usage

```
tidy_patric_meta_data(  
  x,  
  prefer_more_resistant = TRUE,  
  as_ab = TRUE,  
  filter_abx = NULL  
)
```

Arguments

x	PATRIC database loaded using MIC::load_patric_db
prefer_more_resistant	High MICs, narrow zones, or resistant phenotypes will be preferred where multiple reported for the same isolate
as_ab	convert antibiotics to AMR::ab class (column names are antibiotic codes)
filter_abx	filter antibiotics of interest, provided as a vector of antibiotics character names/codes, or ideally, as AMR::ab classes, created using AMR::as.ab

Value

Tidy data, with antimicrobials in wide format, column names describing methodology ("mic_", "disk_", "pheno_"). S3 class "tidy_patric_db".

Examples

```
db <- data.frame(genome_id = 1,  
  genome_name = "E. coli",  
  antibiotic = "amoxicillin",  
  measurement = 2.0,  
  measurement_unit = "mg/L",  
  laboratory_typing_method = "Agar dilution",  
  resistant_phenotype = "R")  
db <- load_patric_db(db)  
tidy_patric_meta_data(db)
```

train_test_filesystem *Organise files into a train-test filesystem*

Description

Organise files into a train-test filesystem

Usage

```
train_test_filesystem(
  path_to_files,
  file_ext,
  split = 0.8,
  train_folder = "train",
  test_folder = "test",
  shuffle = TRUE,
  overwrite = FALSE
)
```

Arguments

path_to_files	directory containing files
file_ext	file extension to filter
split	training data split
train_folder	name of training folder (subdirectory), will be created if does not exist
test_folder	name of testing folder (subdirectory), will be created if does not exist
shuffle	randomise files when splitting (if FALSE, files will be sorted by filename prior to splitting)
overwrite	force overwrite of files that already exist

Value

named vector of train and test directories

Examples

```
set.seed(123)
# create 10 random DNA files
tmp_dir <- tempdir()
# remove any existing .fna files
file.remove(
  list.files(tmp_dir, pattern = "*.fna", full.names = TRUE)
)

for (i in 1:10) {
  writeLines(paste0(">", i, "\n", paste0(sample(c("A", "T", "C", "G"),
    100, replace = TRUE), collapse = "")), file.path(tmp_dir, paste0(i, ".fna")))
}

# split files into train and test directories
paths <- train_test_filesystem(tmp_dir,
  file_ext = "fna",
  split = 0.8,
  shuffle = TRUE,
  overwrite = TRUE)
```

```
list.files(paths[["train"]])
list.files(paths[["test"]])
```

unsqueezed_index_to_str

Get str conversion of unsqueezed kmer using index

Description

Get str conversion of unsqueezed kmer using index

Usage

```
unsqueezed_index_to_str(x, k, starting_index = 1L)
```

Arguments

x integer vector of kmer indices
k kmer length
starting_index starting index (libsvm is usually indexed starting at 1)

Value

vector of unsqueezed kmer strings

Examples

```
unsqueezed_index_to_str(2, k = 3)
```

unsqueezed_mers

Generates all permutations of unsqueezed kmers

Description

Generates all permutations of unsqueezed kmers

Usage

```
unsqueezed_mers(k = 3L)
```

Arguments

k kmer length

Value

vector of unsqueezed kmers

Examples

```
unsqueezed_mers(3)
```

```
xgb.cv.lowmem
```

```
Low memory cross-validation wrapper for XGBoost
```

Description

This function performs similar operations to `xgboost::xgb.cv`, but with the operations performed in a memory efficient manner. Unlike `xgboost::xgb.cv`, this version does not load all folds into memory from the start. Rather it loads each fold into memory sequentially, and trains each fold using `xgboost::xgb.train`. This allows larger datasets to be cross-validated.

The main disadvantage of this function is that it is not possible to perform early stopping based the results of all folds. The function does accept an early stopping argument, but this is applied to each fold separately. This means that different folds can (and should be expected to) train for a different number of rounds.

This function also allows for a train-test split (as opposed to multiple) folds. This is done by providing a value of less than 1 to `nfold`, or a list of 1 fold to folds. This is not possible with `xgboost::xgb.cv`, but can be desirable if there is downstream processing that depends on an `xgb.cv.synchronous` object (which is the return object of both this function and `xgboost::xgb.cv`).

Otherwise, where possible this function tries to return the same data structure as `xgboost::xgb.cv`, with the exception of callbacks (not supported as a field within the return object). To save models, use the `save_models` argument, rather than the `cb.cv.predict(save_models = TRUE)` callback.

Usage

```
xgb.cv.lowmem(
  params = list(),
  data,
  nrounds,
  nfold,
  label = NULL,
  missing = NA,
  prediction = FALSE,
  metrics = list(),
  obj = NULL,
  feval = NULL,
  stratified = TRUE,
  folds = NULL,
  train_folds = NULL,
  verbose = 1,
  print_every_n = 1L,
  early_stopping_rounds = NULL,
  maximize = NULL,
  save_models = FALSE,
  ...
)
```

Arguments

params	parameters for xgboost
data	DMatrix or matrix
nrounds	number of training rounds
nfold	number of folds, or if < 1 then the proportion will be used as the training split in a train-test split
label	data labels (alternatively provide with DMatrix)
missing	handling of missing data (see xgb.cv)
prediction	return predictions
metrics	evaluation metrics
obj	custom objective function
feval	custom evaluation function
stratified	whether to use stratified folds
folds	custom folds
train_folds	custom train folds
verbose	verbosity level
print_every_n	print every n iterations
early_stopping_rounds	early stopping rounds (applied to each fold)
maximize	whether to maximize the evaluation metric
save_models	whether to save the models
...	additional arguments passed to xgb.train

Value

xgb.cv.synchronous object

Examples

```
train <- list(data = matrix(rnorm(20), ncol = 2),
              label = rbinom(10, 1, 0.5))
dtrain <- xgboost::xgb.DMatrix(train$data, label = train$label, nthread = 1)
cv <- xgb.cv.lowmem(data = dtrain,
                   params = list(objective = "binary:logistic"),
                   nrounds = 2,
                   nfold = 3,
                   prediction = TRUE,
                   nthread = 1)

cv
```


Index

- * **datasets**
 - ecoffs, [8](#)
 - example_mics, [9](#)
- bias, [3](#)
- clean_raw_mic, [3](#)
- combined_file_system, [4](#)
- compare_mic, [5](#)
- compare_sir, [6](#)
- download_patric_db, [7](#)
- ecoffs, [8](#)
- essential_agreement, [8](#)
- example_mics, [9](#)
- fill_dilution_levels, [10](#)
- force_mic, [11](#)
- genome_to_libsvm, [13](#)
- genome_to_libsvm(), [13](#)
- genomes_to_kmer_libsvm, [12](#)
- genomes_to_kmer_libsvm(), [14](#)
- get_mic, [14](#)
- kmers, [15](#)
- load_patric_db, [17](#)
- mic_censor, [17](#)
- mic_r_breakpoint, [19](#)
- mic_range, [18](#)
- mic_s_breakpoint, [20](#)
- mic_uncensor, [20](#)
- move_files, [22](#)
- plot.mic_validation, [23](#)
- print.mic_validation, [24](#)
- print.mic_validation_summary, [25](#)
- pull_PATRIC_genomes, [25](#)
- qc_in_range, [26](#)
- qc_on_target, [27](#)
- replace_multiple_slashes, [28](#)
- reverse_complement, [29](#)
- split_and_combine_files, [29](#)
- split_and_combine_files(), [12](#)
- squeezed_index_to_str, [31](#)
- squeezed_mers, [32](#)
- standardise_mic, [32](#)
- summary.mic_validation, [34](#)
- table, [34](#)
- tidy_patric_meta_data, [35](#)
- train_test_filesystem, [36](#)
- unsqueezed_index_to_str, [38](#)
- unsqueezed_mers, [38](#)
- with_progress, [12](#)
- xgb.cv.lowmem, [39](#)