Package 'zoomerjoin'

August 27, 2026

Title Superlatively Fast Fuzzy Joins

Version 0.2.4

Description Empowers users to fuzzily-merge data frames with millions or tens of millions of rows in minutes with low memory usage. The package uses the locality sensitive hashing algorithms developed by Datar, Immorlica, Indyk and Mirrokni (2004) <doi:10.1145/997817.997857>, and Broder (1998) <doi:10.1109/SEQUEN.1997.666900> to avoid having to compare every pair of records in each dataset, resulting in fuzzy-merges that finish in linear time.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

SystemRequirements Cargo (>= 1.56) (Rust's package manager), rustc (>= 1.70)

Imports collapse, dplyr, tibble, tidyr, rlang,

Suggests babynames, fuzzyjoin, covr, igraph, knitr, microbenchmark, profmem, purrr, rmarkdown, stringdist, testthat (>= 3.0.0), tidyverse, vdiffr

Config/testthat/edition 3

URL https://archive.beniamino.org/zoomerjoin/, https://github.com/beniaminogreen/zoomerjoin

BugReports https://github.com/beniaminogreen/zoomerjoin/issues

VignetteBuilder knitr

Depends R (>= 4.2)

LazyData true

LazyDataCompression xz

Config/rextendr/version 0.4.2

NeedsCompilation yes

Author Beniamino Green [aut, cre, cph], Etienne Bacher [ctb] (ORCID: <https://orcid.org/0000-0002-9271-5075>), The authors of the dependency Rust crates [ctb, cph] (see inst/AUTHORS file for details)

Maintainer Beniamino Green <beniamino.green@yale.edu>

Repository CRAN

Date/Publication 2026-08-27 05:10:02 UTC

Contents

dime_data . . . 2
em_link . . . 3
euclidean_anti_join . . . 4
euclidean_curve . . . 7
euclidean_probability . . . 7
fuzzy_join_core . . . 8
hamming_distance . . . 9
hamming_inner_join . . . 10
hamming_probability . . . 13
jaccard_curve . . . 13
jaccard_hyper_grid_search . . . 14
jaccard_inner_join . . . 15
jaccard_probability . . . 18
jaccard_similarity . . . 19
jaccard_string_group . . . 20

Index **22**

`dime_data` *Donors from DIME Database*

Description

A set of donor names from the Database on Ideology, Money in Politics, and Elections (DIME). This dataset was used as a benchmark in the 2021 APSR paper Adaptive Fuzzy String Matching: How to Merge Datasets with Only One (Messy) Identifying Field by Aaron R. Kaufman and Aja Klevs, the dataset in this package is a subset of the data from the replication archive of that paper. The full dataset can be found in the paper's replication materials here: doi:10.7910/DVN/4031UL.

Usage

```
dime_data
```

Format

`dime_data`**:**

A data frame with 10,000 rows and 2 columns:

id Numeric ID / Row Number

x Donor Name ...

#' @source https://www.who.int/teams/global-tuberculosis-programme/data

Author(s)

Adam Bonica

References

doi:10.7910/DVN/4031UL

`em_link` *Fit a Probabilistic Matching Model using Naive Bayes + E.M.*

Description

A Rust implementation of the Naive Bayes / Fellegi-Sunter model of record linkage as detailed in the article "Using a Probabilistic Model to Assist Merging of Large-Scale Administrative Records" by Enamorado, Fifield and Imai (2019). Takes an integer matrix describing the similarities between each possible pair of observations, and a vector of initial guesses of the probability each pair is a match (these can either be set from domain knowledge, or one can hand-label a subset of the data and leave the rest as p=.5). Iteratively refines these guesses using the Expectation Maximization algorithm until an optima is reached. for more details, see doi:10.1017/S0003055418000783.

Usage

```
em_link(X, g, tol = 10^-6, max_iter = 10^3)
```

Arguments

`X`	an integer matrix of similarities. Must go from 0 (the most disagreement) to the maximum without any "gaps" or unused levels. As an example, a column with values 0,1,2,3 is a valid column, but 0,1,2,4 is not as three is omitted
`g`	a vector of initial guesses that are iteratively improved using the EM algorithm (my personal approach is to guess at logistic regression coefficients and use them to create the intitial probability guesses). This is chosen to avoid the model getting stuck in a local optimum, and to avoid the problem of label-switching, where the labels for matches and non-matches are reversed.
`tol`	tolerance in the sense of the infinity norm. i.e. how close the parameters have to be between iterations before the EM algorithm terminates.
`max_iter`	iterations after which the algorithm will error out if it has not converged.

Value

a vector of probabilities representing the posterior probability each record pair is a match.

Examples

```
inv_logit <- function(x) {
  exp(x) / (1 + exp(x))
}
n <- 10^6
d <- 1:n %% 5 == 0
X <- cbind(
  as.integer(ifelse(d, runif(n) < .8, runif(n) < .2)),
  as.integer(ifelse(d, runif(n) < .9, runif(n) < .2)),
  as.integer(ifelse(d, runif(n) < .7, runif(n) < .2)),
  as.integer(ifelse(d, runif(n) < .6, runif(n) < .2)),
  as.integer(ifelse(d, runif(n) < .5, runif(n) < .2)),
  as.integer(ifelse(d, runif(n) < .1, runif(n) < .9)),
  as.integer(ifelse(d, runif(n) < .1, runif(n) < .9)),
  as.integer(ifelse(d, runif(n) < .8, runif(n) < .01))
)

# inital guess at class assignments based on # a hypothetical logistic
# regression. Should be based on domain knowledge, or a handful of hand-coded
# observations.

x_sum <- rowSums(X)
g <- inv_logit((x_sum - mean(x_sum)) / sd(x_sum))

out <- em_link(X, g, tol = .0001, max_iter = 100)
```

---

euclidean_anti_join *Fuzzy joins for Euclidean distance using Locality Sensitive Hashing*

Description

Fuzzy joins for Euclidean distance using Locality Sensitive Hashing

Usage

```
euclidean_anti_join(
  a,
  b,
  by = NULL,
  threshold = 1,
  n_bands = 30,
  band_width = 5,
  r = 0.5,
  progress = FALSE,
  nthread = NULL
)
```

```
euclidean_inner_join(
  a,
  b,
  by = NULL,
  threshold = 1,
  n_bands = 30,
  band_width = 5,
  r = 0.5,
  progress = FALSE,
  nthread = NULL
)

euclidean_left_join(
  a,
  b,
  by = NULL,
  threshold = 1,
  n_bands = 30,
  band_width = 5,
  r = 0.5,
  progress = FALSE,
  nthread = NULL
)

euclidean_right_join(
  a,
  b,
  by = NULL,
  threshold = 1,
  n_bands = 30,
  band_width = 5,
  r = 0.5,
  progress = FALSE,
  nthread = NULL
)

euclidean_full_join(
  a,
  b,
  by = NULL,
  threshold = 1,
  n_bands = 30,
  band_width = 5,
  r = 0.5,
  progress = FALSE,
  nthread = NULL
)
```

Arguments

a, b	The two dataframes to join.
by	A named vector indicating which columns to join on. Format should be the same as dplyr: `by = c("column_name_in_df_a" = "column_name_in_df_b")`, but two columns must be specified in each dataset (x column and y column). Specification made with `dplyr::join_by()` are also accepted.
threshold	The distance threshold below which units should be considered a match. Note that contrary to Jaccard joins, this value is about the distance and not the similarity. Therefore, a lower value means a higher similarity.
n_bands	The number of bands used in the minihash algorithm (default is 40). Use this in conjunction with the `band_width` to determine the performance of the hashing. The default settings are for a (.2, .8, .001, .999)-sensitive hash i.e. that pairs with a similarity of less than .2 have a >.1% chance of being compared, while pairs with a similarity of greater than .8 have a >99.9% chance of being compared.
band_width	The length of each band used in the minihashing algorithm (default is 8) Use this in conjunction with the `n_bands` to determine the performance of the hashing. The default settings are for a (.2, .8, .001, .999)-sensitive hash i.e. that pairs with a similarity of less than .2 have a >.1% chance of being compared, while pairs with a similarity of greater than .8 have a >99.9% chance of being compared.
r	Hyperparameter used to govern the sensitivity of the locality sensitive hash. Corresponds to the width of the hash bucket in the LSH algorithm. Increasing values of `r` mean more hash collisions and higher sensitivity (fewer false-negatives) at the cost of lower specificity (more false-positives and longer run time). For more information, see the description in doi:10.1145/997817.997857.
progress	Set to `TRUE` to print progress.
nthread	Maximum number of threads to use. If `NULL` (default), Rayon's global thread pool is used, which typically uses all logical CPU cores available.

Value

A tibble fuzzily-joined on the basis of the variables in `by`. Tries to adhere to the same standards as the dplyr-joins, and uses the same logical joining patterns (i.e. inner-join joins and keeps only observations in both datasets).

References

Datar, Mayur, Nicole Immorlica, Pitor Indyk, and Vahab Mirrokni. "Locality-Sensitive Hashing Scheme Based on p-Stable Distributions" SCG '04: Proceedings of the twentieth annual symposium on Computational geometry (2004): 253-262

Examples

```
n <- 10

# Build two matrices that have close values
X_1 <- matrix(c(seq(0, 1, 1 / (n - 1)), seq(0, 1, 1 / (n - 1))), nrow = n)
X_2 <- X_1 + .0000001
```

```
X_1 <- as.data.frame(X_1)
X_2 <- as.data.frame(X_2)

X_1$id_1 <- 1:n
X_2$id_2 <- 1:n

# only keep observations that have a match
euclidean_inner_join(X_1, X_2, by = c("V1", "V2"), threshold = .00005)

# keep all observations from X_1, regardless of whether they have a match
euclidean_inner_join(X_1, X_2, by = c("V1", "V2"), threshold = .00005)
```

`euclidean_curve` *Plot S-Curve for a LSH with given hyperparameters*

Description

Plot S-Curve for a LSH with given hyperparameters

Usage

```
euclidean_curve(n_bands, band_width, r, up_to = 100)
```

Arguments

`n_bands`	The number of LSH bands calculated
`band_width`	The number of hashes in each band
`r`	the "r" hyperparameter used to govern the sensitivity of the hash.
`up_to`	the right extent of the x axis.

Value

A plot showing the probability a pair is proposed as a match, given the Jaccard similarity of the two items.

`euclidean_probability` *Find Probability of Match Based on Similarity*

Description

Find Probability of Match Based on Similarity

Usage

```
euclidean_probability(distance, n_bands, band_width, r)
```

Arguments

distance	the euclidian distance between the two vectors you want to compare.
n_bands	The number of LSH bands used in hashing.
band_width	The number of hashes in each band.
r	the "r" hyperparameter used to govern the sensitivity of the hash.

Value

a decimal number giving the proability that the two items will be returned as a candidate pair from the minihash algorithm.

---

fuzzy_join_core *Perform a Fuzzy-Join With an Arbitrary Distance Metric*

Description

Code used by zoomerjoin to perform dplyr-style joins. Users wishing to write their own joining functions can extend zoomerjoin's functionality by writing joining functions to use with `fuzzy_join_core`.

Usage

```
fuzzy_join_core(
  a,
  b,
  by,
  join_func,
  mode,
  block_by = NULL,
  similarity_column = NULL,
  ...
)
```

Arguments

a, b	The two dataframes to join.
by	A named vector indicating which columns to join on. Format should be the same as dplyr: `by = c("column_name_in_df_a" = "column_name_in_df_b")`, but two columns must be specified in each dataset (x column and y column). Specification made with `dplyr::join_by()` are also accepted.
join_func	the joining function responsible for performing the join.
mode	the dplyr-style type of join you want to perform
block_by	A named vector indicating which columns to 'block' (perform exact joining) on. Format should be the same as dplyr: `by = c("column_name_in_df_a" = "column_name_in_df_b")`, but two columns must be specified in each dataset (x column and y column). Specification made with `dplyr::join_by()` are also accepted.

similarity_column
: An optional character vector. If provided, the data frame will contain a column with this name giving the similarity between the two fields. Extra column will not be present if anti-joining.

...
: Other parameters to be passed to the joining function

---

hamming_distance *Calculate Hamming distance of two character vectors*

Description

Calculate Hamming distance of two character vectors

Usage

```
hamming_distance(a, b, nthread = NULL)
```

Arguments

a	the first character vector
b	the first character vector
nthread	Maximum number of threads to use. If NULL (default), Rayon's global thread pool is used, which typically uses all logical CPU cores available.

Value

a vector of hamming similarities of the strings

Examples

```
hamming_distance(
  c("ACGTCGATGACGTGATGCGTAGCGTA", "ACGTCGATGTGCTCTCGTCGATCTAC"),
  c("ACGTCGACGACGTGATGCGCAGCGTA", "ACGTCGATGGGGTCTCGTCGATCTAC")
)
```

hamming_inner_join *Fuzzy joins for Hamming distance using Locality Sensitive Hashing*

Description

Find similar rows between two tables using the hamming distance. The hamming distance is equal to the number characters two strings differ by, or is equal to infinity if two strings are of different lengths

Usage

```
hamming_inner_join(
  a,
  b,
  by = NULL,
  n_bands = 100,
  band_width = 8,
  threshold = 2,
  progress = FALSE,
  clean = FALSE,
  similarity_column = NULL,
  nthread = NULL
)

hamming_anti_join(
  a,
  b,
  by = NULL,
  n_bands = 100,
  band_width = 100,
  threshold = 2,
  progress = FALSE,
  clean = FALSE,
  similarity_column = NULL,
  nthread = NULL
)

hamming_left_join(
  a,
  b,
  by = NULL,
  n_bands = 100,
  band_width = 100,
  threshold = 2,
  progress = FALSE,
  clean = FALSE,
  similarity_column = NULL,
```

```
  nthread = NULL
)

hamming_right_join(
  a,
  b,
  by = NULL,
  n_bands = 100,
  band_width = 100,
  threshold = 2,
  progress = FALSE,
  clean = FALSE,
  similarity_column = NULL,
  nthread = NULL
)

hamming_full_join(
  a,
  b,
  by = NULL,
  n_bands = 100,
  band_width = 100,
  threshold = 2,
  progress = FALSE,
  clean = FALSE,
  similarity_column = NULL,
  nthread = NULL
)
```

Arguments

a, b	The two dataframes to join.
by	A named vector indicating which columns to join on. Format should be the same as dplyr: by = c("column_name_in_df_a" = "column_name_in_df_b"), but two columns must be specified in each dataset (x column and y column). Specification made with dplyr::join_by() are also accepted.
n_bands	The number of bands used in the locality sensitive hashing algorithm (default is 100). Use this in conjunction with the band_width to determine the performance of the hashing. Generally speaking, a higher number of bands leads to greater recall at the cost of higher runtime.
band_width	The length of each band used in the minihashing algorithm (default is 8). Use this in conjunction with the n_bands to determine the performance of the hashing. Generally speaking a wider number of bands decreases the number of false positives, decreasing runtime at the cost of lower sensitivity (true matches are less likely to be found).
threshold	The Hamming distance threshold below which two strings should be considered a match. A distance of zero corresponds to complete equality between strings,

while a distance of 'x' between two strings means that 'x' substitutions must be made to transform one string into the other.

`progress` Set to `TRUE` to print progress.

`clean` Should the strings that you fuzzy join on be cleaned (coerced to lower-case, stripped of punctuation and spaces)? Default is `FALSE`.

`similarity_column`
An optional character vector. If provided, the data frame will contain a column with this name giving the Hamming distance between the two fields. Extra column will not be present if anti-joining.

`nthread` Maximum number of threads to use. If `NULL` (default), Rayon's global thread pool is used, which typically uses all logical CPU cores available.

Value

A tibble fuzzily-joined on the basis of the variables in `by`. Tries to adhere to the same standards as the dplyr-joins, and uses the same logical joining patterns (i.e. inner-join joins and keeps only observations in both datasets).

Examples

```
if (requireNamespace("babynames", quietly = TRUE)) {
  baby_names <- data.frame(
    name = tolower(unique(babynames::babynames$name))[1:500]
  )

  baby_names_mispelled <- data.frame(
    name_mispelled = gsub("[aeiouy]", "x", baby_names$name)
  )

  hamming_inner_join(
    baby_names,
    baby_names_mispelled,
    by = c("name" = "name_mispelled"),
    threshold = 3,
    n_bands = 150,
    band_width = 10,
    clean = FALSE
  )

  hamming_left_join(
    baby_names,
    baby_names_mispelled,
    by = c("name" = "name_mispelled"),
    threshold = 3,
    n_bands = 150,
    band_width = 10
  )
}
```

hamming_probability *Find Probability of Match Based on Similarity*

Description

Find Probability of Match Based on Similarity

Usage

```
hamming_probability(distance, input_length, n_bands, band_width)
```

Arguments

distance	The hamming distance of the two strings you want to compare
input_length	the length (number of characters) of the input strings you want to calculate.
n_bands	The number of LSH bands used in hashing.
band_width	The number of hashes in each band.

Value

A decimal number giving the probability that the two items will be returned as a candidate pair from the lsh algotithm.

jaccard_curve *Plot S-Curve for a LSH with given hyperparameters*

Description

Plot S-Curve for a LSH with given hyperparameters

Usage

```
jaccard_curve(n_bands, band_width)
```

Arguments

n_bands	The number of LSH bands calculated
band_width	The number of hashes in each band

Value

A plot showing the probability a pair is proposed as a match, given the Jaccard similarity of the two items.

Examples

```
# Plot the probability two pairs will be matched as a function of their
# jaccard similarity, given the hyperparameters n_bands and band_width.
jaccard_curve(40, 6)
```

`jaccard_hyper_grid_search` *Help Choose the Appropriate LSH Hyperparameters*

Description

Runs a grid search to find the hyperparameters that will achieve an (s1,s2,p1,p2)-sensitive locality sensitive hash. A locality sensitive hash can be called (s1,s2,p1,p2)-sensitive if to strings with a similarity less than s1 have a less than p1 chance of being compared, while two strings with similarity s2 have a greater than p2 chance of being compared. As an example, a (.1,.7,.001,.999)-sensitive LSH means that strings with similarity less than .1 will have a .1% chance of being compared, while strings with .7 similarity have a 99.9% chance of being compared.

Usage

```
jaccard_hyper_grid_search(s1 = 0.1, s2 = 0.7, p1 = 0.001, p2 = 0.999)
```

Arguments

s1	the s1 parameter (the first similaity).
s2	the s2 parameter (the second similarity, must be greater than s1).
p1	the p1 parameter (the first probability).
p2	the p2 parameter (the second probability, must be greater than p1).

Value

a named vector with the hyperparameters that will meet the LSH criteria, while reducing runitme.

Examples

```
# Help me find the parameters that will minimize runtime while ensuring that
# two strings with similarity .1 will be compared less than .1% of the time,
# strings with .8 similaity will have a 99.95% chance of being compared:
jaccard_hyper_grid_search(.1, .9, .001, .995)
```

jaccard_inner_join *Fuzzy joins for Jaccard distance using MinHash*

Description

Fuzzy joins for Jaccard distance using MinHash

Usage

```
jaccard_inner_join(
  a,
  b,
  by = NULL,
  block_by = NULL,
  n_gram_width = 2,
  n_bands = 50,
  band_width = 8,
  threshold = 0.7,
  progress = FALSE,
  clean = FALSE,
  similarity_column = NULL,
  nthread = NULL
)

jaccard_anti_join(
  a,
  b,
  by = NULL,
  block_by = NULL,
  n_gram_width = 2,
  n_bands = 50,
  band_width = 8,
  threshold = 0.7,
  progress = FALSE,
  clean = FALSE,
  similarity_column = NULL,
  nthread = NULL
)

jaccard_left_join(
  a,
  b,
  by = NULL,
  block_by = NULL,
  n_gram_width = 2,
  n_bands = 50,
  band_width = 8,
```

```
  threshold = 0.7,
  progress = FALSE,
  clean = FALSE,
  similarity_column = NULL,
  nthread = NULL
)

jaccard_right_join(
  a,
  b,
  by = NULL,
  block_by = NULL,
  n_gram_width = 2,
  n_bands = 50,
  band_width = 8,
  threshold = 0.7,
  progress = FALSE,
  clean = FALSE,
  similarity_column = NULL,
  nthread = NULL
)

jaccard_full_join(
  a,
  b,
  by = NULL,
  block_by = NULL,
  n_gram_width = 2,
  n_bands = 50,
  band_width = 8,
  threshold = 0.7,
  progress = FALSE,
  clean = FALSE,
  similarity_column = NULL,
  nthread = NULL
)
```

Arguments

a, b	The two dataframes to join.
by	A named vector indicating which columns to join on. Format should be the same as dplyr: by = c("column_name_in_df_a" = "column_name_in_df_b"), but two columns must be specified in each dataset (x column and y column). Specification made with dplyr::join_by() are also accepted.
block_by	A named vector indicating which column to block on, such that rows that disagree on this field cannot be considered a match. Format should be the same as dplyr: by = c("column_name_in_df_a" = "column_name_in_df_b")
n_gram_width	The length of the n_grams used in calculating the Jaccard similarity. For best

performance, I set this large enough that the chance any string has a specific n_gram is low (i.e. `n_gram_width` = 2 or 3 when matching on first names, 5 or 6 when matching on entire sentences).

`n_bands` The number of bands used in the minihash algorithm (default is 40). Use this in conjunction with the `band_width` to determine the performance of the hashing. The default settings are for a (.2, .8, .001, .999)-sensitive hash i.e. that pairs with a similarity of less than .2 have a >.1% chance of being compared, while pairs with a similarity of greater than .8 have a >99.9% chance of being compared.

`band_width` The length of each band used in the minihashing algorithm (default is 8) Use this in conjunction with the `n_bands` to determine the performance of the hashing. The default settings are for a (.2, .8, .001, .999)-sensitive hash i.e. that pairs with a similarity of less than .2 have a >.1% chance of being compared, while pairs with a similarity of greater than .8 have a >99.9% chance of being compared.

`threshold` The Jaccard similarity threshold above which two strings should be considered a match (default is .95). The similarity is equal to 1 - the Jaccard distance between the two strings, so 1 implies the strings are identical, while a similarity of zero implies the strings are completely dissimilar.

`progress` Set to `TRUE` to print progress.

`clean` Should the strings that you fuzzy join on be cleaned (coerced to lower-case, stripped of punctuation and spaces)? Default is `FALSE`.

`similarity_column`
An optional character vector. If provided, the data frame will contain a column with this name giving the Jaccard similarity between the two fields. Extra column will not be present if anti-joining.

`nthread` Maximum number of threads to use. If `NULL` (default), Rayon's global thread pool is used, which typically uses all logical CPU cores available.

Value

A tibble fuzzily-joined on the basis of the variables in `by`. Tries to adhere to the same standards as the dplyr-joins, and uses the same logical joining patterns (i.e. inner-join joins and keeps only observations in both datasets).

Examples

```
# load baby names data
# install.packages("babynames")
if (requireNamespace("babynames", quietly = TRUE)) {
  baby_names <- data.frame(
    name = tolower(unique(babynames::babynames$name))[1:500]
  )

  baby_names_sans_vowels <- data.frame(
    name_wo_vowels = gsub("[aeiouy]", "", baby_names$name)
  )

  # Check the probability two pairs of strings with similarity .8 will be
  # matched with a band width of 8 and 30 bands using the `jaccard_probability()`
```

```
  # function:
  jaccard_probability(.8, 30, 8)

  # Run the join and only keep rows that have a match:
  jaccard_inner_join(
    baby_names,
    baby_names_sans_vowels,
    by = c("name" = "name_wo_vowels"),
    threshold = .8,
    n_bands = 20,
    band_width = 6,
    n_gram_width = 1,
    clean = FALSE # default
  )

  # Run the join and keep all rows from the first dataset, regardless of whether
  # they have a match:
  jaccard_left_join(
    baby_names,
    baby_names_sans_vowels,
    by = c("name" = "name_wo_vowels"),
    threshold = .8,
    n_bands = 20,
    band_width = 6,
    n_gram_width = 1
  )
}
```

`jaccard_probability` *Find Probability of Match Based on Similarity*

Description

This is a port of the lsh_probability function from the textreuse package, with arguments changed to reflect the hyperparameters in this package. It gives the probability that two strings of jaccard similarity `similarity` will be matched, given the chosen bandwidth and number of bands.

Usage

```
jaccard_probability(similarity, n_bands, band_width)
```

Arguments

`similarity`	the similarity of the two strings you want to compare
`n_bands`	The number of LSH bands used in hashing.
`band_width`	The number of hashes in each band.

Value

a decimal number giving the probability that the two items will be returned as a candidate pair from the minhash algorithm.

Examples

```
# Find the probability two pairs will be matched given they have a
# jaccard_similarity of .8, band width of 5, and 50 bands:
jaccard_probability(.8, n_bands = 50, band_width = 5)
```

---

jaccard_similarity *Calculate Jaccard Similarity of two character vectors*

Description

Calculate Jaccard Similarity of two character vectors

Usage

```
jaccard_similarity(a, b, ngram_width = 2, nthread = NULL)
```

Arguments

a	the first character vector
b	the first character vector
ngram_width	the length of the shingles / ngrams used in the similarity calculation
nthread	Maximum number of threads to use. If NULL (default), Rayon's global thread pool is used, which typically uses all logical CPU cores available.

Value

a vector of jaccard similarities of the strings

Examples

```
jaccard_similarity(
  c("the quick brown fox", "jumped over the lazy dog"),
  c("the quck bron fx", "jumped over hte lazy dog")
)
```

`jaccard_string_group` *Fuzzy String Grouping Using Minhashing*

Description

Performs fuzzy string grouping in which similar strings are assigned to the same group. Uses the `cluster_fast_greedy()` community detection algorithm from the `igraph` package to create the groups. Must have igraph installed in order to use this function.

Usage

```
jaccard_string_group(
  string,
  n_gram_width = 2,
  n_bands = 45,
  band_width = 8,
  threshold = 0.7,
  progress = FALSE,
  nthread = NULL
)
```

Arguments

`string`	a character you wish to perform entity resolution on.
`n_gram_width`	the length of the n_grams used in calculating the jaccard similarity. For best performance, I set this large enough that the chance any string has a specific n_gram is low (i.e. `n_gram_width` = 2 or 3 when matching on first names, 5 or 6 when matching on entire sentences).
`n_bands`	the number of bands used in the minihash algorithm (default is 40). Use this in conjunction with the `band_width` to determine the performance of the hashing. The default settings are for a (.2,.8,.001,.999)-sensitive hash i.e. that pairs with a similarity of less than .2 have a >.1% chance of being compared, while pairs with a similarity of greater than .8 have a >99.9% chance of being compared.
`band_width`	the length of each band used in the minihashing algorithm (default is 8) Use this in conjunction with the `n_bands` to determine the performance of the hashing. The default settings are for a (.2,.8,.001,.999)-sensitive hash i.e. that pairs with a similarity of less than .2 have a >.1% chance of being compared, while pairs with a similarity of greater than .8 have a >99.9% chance of being compared.
`threshold`	the jaccard similarity threshold above which two strings should be considered a match (default is .95). The similarity is euqal to 1 • the jaccard distance between the two strings, so 1 implies the strings are identical, while a similarity of zero implies the strings are completely dissimilar.
`progress`	set to true to report progress of the algorithm
`nthread`	Maximum number of threads to use. If `NULL` (default), Rayon's global thread pool is used, which typically uses all logical CPU cores available.

Value

a string vector storing the group of each element in the original input strings. The input vector is grouped so that similar strings belong to the same group, which is given a standardized name.

Examples

```
if (requireNamespace("igraph", quietly = TRUE)) {
  string <- c(
    "beniamino", "jack", "benjamin", "beniamin",
    "jacky", "giacomo", "gaicomo"
  )
  jaccard_string_group(
    string,
    threshold = 0.2,
    n_bands = 90,
    n_gram_width = 1
  )
}
```

Index

∗ **datasets**
    dime_data, 2

dime_data, 2

em_link, 3
euclidean_anti_join, 4
euclidean_curve, 7
euclidean_full_join
        *(*euclidean_anti_join*)*, 4
euclidean_inner_join
        *(*euclidean_anti_join*)*, 4
euclidean_left_join
        *(*euclidean_anti_join*)*, 4
euclidean_probability, 7
euclidean_right_join
        *(*euclidean_anti_join*)*, 4

fuzzy_join_core, 8

hamming_anti_join *(*hamming_inner_join*)*, 10
hamming_distance, 9
hamming_full_join *(*hamming_inner_join*)*, 10
hamming_inner_join, 10
hamming_left_join *(*hamming_inner_join*)*, 10
hamming_probability, 13
hamming_right_join
        *(*hamming_inner_join*)*, 10

jaccard_anti_join *(*jaccard_inner_join*)*, 15
jaccard_curve, 13
jaccard_full_join *(*jaccard_inner_join*)*, 15
jaccard_hyper_grid_search, 14
jaccard_inner_join, 15
jaccard_left_join *(*jaccard_inner_join*)*, 15
jaccard_probability, 18
jaccard_right_join
        *(*jaccard_inner_join*)*, 15
jaccard_similarity, 19
jaccard_string_group, 20