Package: nmatch (via r-universe)

October 24, 2024

Title Fuzzy Matching For Proper Names

Version 0.1.0

Description Tools for comparing sets of proper names, accounting for common types of variation in format and style.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Depends R (>= 2.10)

Imports stringi, stringr, stringdist, dplyr, rlang, purrr, tidyr

Suggests testthat, covr

URL https://github.com/epicentre-msf/nmatch

BugReports https://github.com/epicentre-msf/nmatch/issues

Repository https://epicentre-msf.r-universe.dev

RemoteUrl https://github.com/epicentre-msf/nmatch

RemoteRef HEAD

RemoteSha c6da5b8908df6473a22a9b824f2db6cca22152b9

Contents

dat_hospital *Example hospital datasets containing proper names in different formats*

Description

Example hospital datasets, one from an in-patient department (`dat_ipd`), and the other from an ICU department (`dat_icu`). The datasets contain some common patients but with variation in how the names are written. Note these data are fake – the patient names are simply random combinations of common French names.

Format

Data frames each with two columns:

name_ipd/name_icu patient name

date_ipd/date_icu date of entry to given department

match_eval *Evaluate token match details to determine overall match status*

Description

Evaluate token match details to determine overall match status

Usage

```
match_eval(k_x, k_y, n_match, n_match_crit, ...)
```

Arguments

`k_x`	Integer vector specifying number of tokens in names `x`
`k_y`	Integer vector specifying number of tokens in names `y`
`n_match`	Integer vector specifying number of aligned tokens between `x` and `y` that are matching (i.e. based on argument `dist_max` in `nmatch`)
`n_match_crit`	Minimum number of matching tokens for names `x` and `y` to be considered an overall match
`...`	Additional arguments (not used)

Value

Logical vector indicating whether names `x` and `y` match, based on the token match details provided as arguments

names_ex *Example data with proper names from two different sources*

Description

Example data with proper names from two different sources

Usage

```
names_ex
```

Format

A data.frame with 6 rows and 2 variables, both of class character:

name_source1 name from source 1

name_source2 name from source 2

name_standardize *String standardization*

Description

Standardize strings prior to performing a match, using the following transformations:

1. standardize case (`base::toupper`)
2. remove accents/diacritics (`stringi::stri_trans_general`)
3. replace punctuation characters with whitespace
4. remove extraneous space characters with (`stringr::str_squish`)

Usage

```
name_standardize(x)
```

Arguments

x	a string

Value

The standardized version of `x`

Examples

```
name_standardize("angela_merkel")
name_standardize("QUOIREZ, Fran\U00E7oise D.")
```

`nmatch` *Compare sets of proper names accounting for common types of variation in format and style*

Description

Compare proper names across two sources using string-standardization to account for variation in punctuation, accents, and character case, token-permutation to account for variation in name order, and fuzzy matching to handle alternate spellings. The specific steps are:

1. Standardize strings. The default function is `name_standardize` which removes accents and punctuation, standardizes case, and removes extra whitespace. E.g. "Brontë, Emily J." is standardized to "BRONTE EMILY J".
2. Tokenize standardized names, optionally retaining only tokens larger than a given nchar limit.
3. For each pair of names, calculate string distance between all combinations of tokens, and find the best overall token alignment (i.e. the alignment that minimizes the summed string distance). If two names being compared differ in their number of tokens, the alignment is made with respect to the smaller number of tokens. E.g. If comparing "Angela Dorothea Merkel" to "Merkel Angela", the token "Dorothea" would ultimately be omitted from the best alignment.
4. Summarize the number of tokens in each name, the number of tokens in the best alignment, the number of aligned tokens that match (i.e. string distance less than or equal to the defined threshold), and the summed string distance of the best alignment.
5. Classify overall match status (TRUE/FALSE) based on match details described in (4). By default, two names are considered to be matching if two or more tokens match across names (e.g. "Merkel Angela" matches "Angela Dorothea Merkel"), or if both names consist of only a single token which is matching (e.g. "Beyonce" matches "Beyoncé").

Usage

```
nmatch(
  x,
  y,
  token_split = "[-_[:space:]]+",
  nchar_min = 2L,
  dist_method = "osa",
  dist_max = 1L,
  std = name_standardize,
  ...,
  return_full = FALSE,
  eval_fn = match_eval,
  eval_params = list(n_match_crit = 2)
)
```

Arguments

x, y	Vectors of proper names to compare. Must be of same length.
token_split	Regex pattern to split strings into tokens. Defaults to "[-_[:space:]]+", which splits at each sequence of one more dash, underscore, or space character.
nchar_min	Minimum token size to compare. Defaults to 2L.
dist_method	Method to use for string distance calculation (see stringdist-metrics). Defaults to "osa".
dist_max	Maximum string distance to use to classify matching tokens (i.e. tokens with a string distance less than or equal to dist_max will be considered matching). Defaults to 1L.
std	Function to standardize strings during matching. Defaults to name_standardize. Set to NULL to omit standardization.
...	additional arguments passed to std()
return_full	Logical indicating whether to return data frame with full summary of match details (TRUE), or only a logical vector corresponding to final match status (FALSE). Defaults to FALSE.
eval_fn	Function to determine overall match status. Defaults to match_eval. See section *Custom classification functions* for more details.
eval_params	List of additional arguments passed to eval_fn

Value

If return_full = FALSE (the default), returns a logical vector indicating which elements of x and y are matches.

If return_full = TRUE, returns a tibble-style data frame summarizing the match details, including columns:

- is_match: logical vector indicating overall match status
- k_x: number of tokens in x (excludes tokens smaller than nchar_min)
- k_y: number of tokens in y (excludes tokens smaller than nchar_min)
- k_align: number of aligned tokens (i.e. min(k_x, k_y))
- n_match: number of aligned tokens that match (i.e. distance <= dist_max)
- dist_total: summed string distance across aligned tokens

Examples

```
names1 <- c(
  "Angela Dorothea Merkel",
  "Emmanuel Jean-Michel Fr\u00e9d\u00e9ric Macron",
  "Mette Frederiksen",
  "Katrin Jakobsd\u00f3ttir",
  "Pedro S\u00e1nchez P\u00e9rez-Castej\u00f3n"
)

names2 <- c(
```

```
  "MERKEL, Angela",
  "MACRON, Emmanuel J.-M. F.",
  "FREDERICKSON, Mette",
  "JAKOBSDOTTIR  Kathríne",
  "PEREZ-CASTLEJON, Pedro"
)

# return logical vector specifying which names are matches
nmatch(names1, names2)

# increase the threshold string distance to allow for 'fuzzier' matches
nmatch(names1, names2, dist_max = 2)

# return data frame with full match details
nmatch(names1, names2, return_full = TRUE)

# use a custom function to classify matches
classify_matches <- function(k_align, n_match, dist_total, ...) {
  n_match == k_align & dist_total < 2
}

nmatch(names1, names2, return_full = TRUE, eval_fn = classify_matches)
```

Index