Package 'fedmatch'

January 26, 2018

Title Record linkage functions in R

Version 1.0.0.0

Description Functions for merging two unlinked datasets.
The central function of this package is ``merge_plus",which extends base R merge functionality to include fuzzy string matching, match scoring based on the similarity of common variables between the two datasets, filtering based on a calculated match score or a user-inputed function, match evaluation (see match_evaluate), and safe merge checks.
Other functions include:
-match_evaluate, which produces standard matching statistics including percent matched, and duplicate ratios,
-tier_match, which is a wrapper for merge_plus that allows you match two datasets in sequential tiers with gradually looser parameters,
-calculate_weights, a function that estimates the ability of a common variable to correctly identify a match or a non-match based on the record linkage literature,
-clean_strings, a general string cleaning function optimized for company names.
See ``match_template.R" in the ``examples" folder for a self-contained tutorial on the functionality of this package and template for your own matching program.

Depends R (>= 3.3.1)

License GNU GENERAL PUBLIC LICENSE

Encoding UTF-8

LazyData true

Imports stringdist,
SnowballC,
gtools,

RoxygenNote 6.0.1

R topics documented:

`articles` *articles*

Description

Data.frame with common articles

Usage

```
articles
```

Format

An object of class `data.frame` with 23 rows and 2 columns.

See Also

clean_strings

`calculate_weights` *calculate_weights*

Description

calculate weights for comparison variables based on m and u probabilities estimated from a verified dataset. See https://en.wikipedia.org/wiki/Record_linkage

Usage

```
calculate_weights(data, variables, compare_types = "stringdist",
  suffixes = c(".x", ".y"), non_negative = FALSE)
```

Arguments

data	data.frame. Verified data. Should have all of the variables you want to calculate weights for from both datasets, named the same with data-specific suffixes.
variables	character vector of the variable names of the variables you want to calculate weights for.
suffixes	character vector. Suffixes of of the variables that indicate what data they are from. Default is same as the default for base R merge, c('.x','.y')
non_negative	logical. Do you want to allow negative weights?
compare_type	character vector. One of 'stringdist' (for string variables) 'ratio','difference' (for numerics) 'indicator' (0-1 dummy indicating if the two are the same),'in' (0-1 dummy indicating if data1 is IN data2), and 'substr' (numeric indicating how many digits are the same.)

Value

list with m probabilities, u probabilites, w weights, and settings, the list argument requried as an input for score_settings in merge_plus using the calculate weights.

Examples

```
See match_template.R in examples folder -- end of the file.
```

---

clean_strings *clean_strings*

Description

default string cleaning process for "name_match"

Usage

```
clean_strings(string, sp_char_words = NULL, common_words = NULL,
  remove_char = NULL, remove_words = FALSE, stem = FALSE,
  replace_null = NULL)
```

Arguments

string	character or character vector of strings
sp_char_words	character vector. Data.frame where first column is special characaters and second column is full words.
common_words	data.frame. Data.frame where first column is abbreviations and second column is full words.
remove_char	character vector. string of specific characters (for example, "letters") to be removed

remove_words	logical. If TRUE, removes all abbreviations and replacement words in common_words
stem	logical. If TRUE, words are stemmed

Value

cleaned strings

Examples

```
# basic cleaning example
sample_str = c("Holding Co. A @ B St, 3 ))", "Company B & C 4, Inc.", "make $, inc.")
sample_clean1 = clean_strings(sample_str)

# defining common words with a package database
sample_clean2 = clean_strings(sample_str, common_words=corporate_words)

# dropping common words in a database
sample_clean3 = clean_strings(sample_str, common_words=corporate_words, remove_words=TRUE)

# sunco example
sample_clean4 = clean_strings("co cosuncosunco co co", common_words = cbind(c("co"), c("company")))

# changing special characters to words(Note that @ and & are dropped with punctuation)
drop_char = cbind(c("\$", "\\%"), c("dollar", "percent"))
sample_clean5 = clean_strings(sample_str, sp_char_words = drop_char)
```

---

corporate_words *corporate_words*

Description

Data.frame with common corporate abbreviations in column 1 and corresponding long names in column 2. Useful for cleaning company names for matching.

Usage

```
corporate_words
```

Format

An object of class `data.frame` with 54 rows and 2 columns.

See Also

clean_strings

count_rows *count_rows*

Description

Takes two datasets and a character vector of merge variables, and returns the number of rows of a hypothetical merged dataset, without actually performing the merge. Useful in cases where merge variables may not be unique, and a merge could result in an R-crashingly large dataset.

Usage

```
count_rows(x, y, by, by.x = by, by.y = by)
```

Arguments

x	data.frame. First data to merge
y	data.frame. Second data to merge
merge_ids	character vector. Merge variables

Details

h/t Joris Meys and his Stack Overflow post http://stackoverflow.com/questions/7441188/how-to-efficiently-merge-two-datasets

Value

number of rows of the merged dataset

Examples

```
count_rows(data.frame('id'=c(1,1,1,2,2,3)), data.frame('id'=c(1,1,2,2,3,4)), 'id')
```

data1 *data1*

Description

Some made up data on the top 10 US companies in the Fortune 500. Mock-matched to data2 in examples/match_template.R

Usage

```
data1
```

Format

An object of class `data.frame` with 10 rows and 5 columns.

`data2` *data2*

Description

Some made up data on the top 10 US companies in the Fortune 500. Mock-matched to data1 in examples/match_template.R

Usage

```
data2
```

Format

An object of class `data.frame` with 10 rows and 5 columns.

`duplicate_eliminate` *duplicate_eliminate*

Description

Identifies and eliminates all duplicates including the first instance, if they exist

Usage

```
duplicate_eliminate(data, id_vars, dupout = FALSE, tag = NULL)
```

Arguments

`data`	data.frame
`id_vars`	character vector.
`dupout`	logical. If FALSE, returns dataset without duplicates. If TRUE, returns duplicates.
`tag`	character string. If NULL, return dataset with or without duplicates based on the dupout parameter. Else, return dataset with column "tag" containing duplicate value indictor (0 or 1)

Details

h/t Simon O'Hanlon and his Stack Overflow post http://stackoverflow.com/questions/17352657/using-r-how-can-i-flag-sequential-duplicate-values-in-a-single-column-of-a-data

Value

see dupout

Examples

```
data = data.frame('a'=c(1,1,1,1,2,2,2,2,3), 'b'=c(1,1,2,4,1,2,2,3,1))
duplicate_eliminate(data,c('a','b'))
duplicate_eliminate(data,c('a'))
```

`fund_words` *fund_words*

Description

Data.frame with abbreviations common in the names of financial (i.e. mutual) funds in column 1 and corresponding long names in column 2. Useful for cleaning fund names for matching.

Usage

```
fund_words
```

Format

An object of class `data.frame` with 63 rows and 2 columns.

See Also

clean_strings

`match_evaluate` *match_evaluate*

Description

evaluate a matched dataset

Usage

```
match_evaluate(matches, data1, data2, unique_key_1, unique_key_2,
  suffixes = c(".x", ".y"), tier = NULL, tier_order = NULL,
  aggregate_by = "unique", quality_vars = NULL, dupe_ratio = FALSE)
```

Arguments

matches	data.frame. Merged dataset.
data1	data.frame. First to-merge dataset.
data2	data.frame. Second to-merge dataset.
unique_key_1	character vector. Primary key of data1 that uniquely identifies each row (can be multiple fields)
unique_key_2	character vector. Primary key of data2 that uniquely identifies each row (can be multiple fields)
suffixes	character vector. Mnemonics associated data1 and data2.
tier	character vector. Default=NULL. The variable that defines a tier.
tier_order	character vector. Default=NULL. Variable that defines the order of tiers, if needed.
aggregate_by	character vector. Default='unique'. Variable aggregated to calculate statistics. If equal to 'unique', aggregation is count, if otherwise, sum.
quality_vars	character vector. Variables you want to use to calculate the quality of each tier. Calculates mean.

Value

data.frame. Table describing each tier according to aggregate_by variables and quality_vars variables.

See Also

merge_plus

Examples

```
x = data.frame('id'=1:5,'name'=c('a','b','c','d','d'), 'amount' = 101:105)
y = data.frame('id'=6:10,'name'=c('b','c','d','e','f'), 'amount' = rep(102,5))
matches = merge_plus(x,y,by='name',unique_key_1='id','unique_key_2'='id', matchscore_settings = list('amount'=l
match_evaluate(matches, x, y, 'id.x', 'id.y')
```

---

merge_plus *merge_plus*

Description

merge two datasets, plus.

Usage

```
merge_plus(data1, data2, by = NULL, by.x = by, by.y = by,
  suffixes = c(".x", ".y"), check_merge = TRUE, unique_key_1, unique_key_2,
  match_type = "exact", amatch.args = list(method = "jw", p = 0.1, maxDist =
  0.05, matchNA = FALSE), score_settings = NULL, filter = NULL,
  filter.args = list(), evaluate = match_evaluate, evaluate.args = list())
```

Arguments

data1	data.frame. First to-merge dataset.
data2	data.frame. Second to-merge dataset.
by	character string. Variables to merge on (common across data 1 and data 2). See merge
by.x	character string. Variable to merge on in data1. See merge
by.y	character string. Variable to merge on in data2. See merge
suffixes	character vector with length==2. Suffix to add to like named variables after the merge. See merge
check_merge	logical. Checks that your unique_keys are indeed unique, and prevents merge from running if merge would result in data.frames larger than 5 million rows
unique_key_1	character vector. Primary key of data1 that uniquely identifies each row (can be multiple fields)
unique_key_2	character vector. Primary key of data2 that uniquely identifies each row (can be multiple fields)
score_settings	list. score settings. See vingette matchscore
filter	function or numeric. Filters a merged data1-data2 dataset. If a function, should take in a data.frame (data1 and data2 merged by name1 and name2) and spit out a trimmed verion of the data.frame (fewer rows). Think of this function as applying other conditions to matches, other than a match by name. The first argument of filter should be the data.frame. If numeric, will drop all observations with a matchscore lower than or equal to filter.
filter.args	list. Arguments passed to filter, if a function
evaluate	Function to evalute merge_plus output.
evaluate.args	ist. Arguments passed to evaluate
match_type.	string. If 'exact', match is exact, if 'fuzzy', match is fuzzy.
amatch.args.	additional arguments for amatch, to be used if match_type = 'fuzzy'. Suggested defaults provided. (see amatch, method='jw')

Value

list with matches, filtered matches (if applicable), data1 and data2 minus matches, and match evaluation

See Also

match_evaluate

Examples

```
x = data.frame('id'=1:5,'name'=c('a','b','c','d','d'), 'amount' = 101:105)
y = data.frame('id'=6:10,'name'=c('b','c','d','e','f'), 'amount' = rep(102,5))
merge_plus(x,y,by='name',unique_key_1='id','unique_key_2'='id')
merge_plus(x,y,by='name',unique_key_1='id','unique_key_2'='id',
         matchscore_settings = list('amount'=list('compare_type'='difference',weight=1)),
           filter=.5)
```

`merge_plus_chunk` *merge_plus_chunk*

Description

merge two datasets, plus.

Usage

```
merge_plus_chunk(data1, data2, by = NULL, by.x = by, by.y = by,
  suffixes = c(".x", ".y"), check_merge = TRUE, unique_key_1, unique_key_2,
  match_type = "exact", amatch.args = list(method = "jw", p = 0.1, maxDist =
  0.05, matchNA = FALSE), score_settings = NULL, filter = NULL,
  filter.args = list(), evaluate = match_evaluate, evaluate.args = list(),
  chunk = NULL)
```

Arguments

`data1`	data.frame. First to-merge dataset.
`data2`	data.frame. Second to-merge dataset.
`by`	character string. Variables to merge on (common across data 1 and data 2). See `merge`
`by.x`	character string. Variable to merge on in data1. See `merge`
`by.y`	character string. Variable to merge on in data2. See `merge`
`suffixes`	character vector with length==2. Suffix to add to like named variables after the merge. See `merge`
`check_merge`	logical. Checks that your unique_keys are indeed unique, and prevents merge from running if merge would result in data.frames larger than 5 million rows
`unique_key_1`	character vector. Primary key of data1 that uniquely identifies each row (can be multiple fields)
`unique_key_2`	character vector. Primary key of data2 that uniquely identifies each row (can be multiple fields)
`score_settings`	list. score settings. See vingette matchscore

filter	function or numeric. Filters a merged data1-data2 dataset. If a function, should take in a data.frame (data1 and data2 merged by name1 and name2) and spit out a trimmed verion of the data.frame (fewer rows). Think of this function as applying other conditions to matches, other than a match by name. The first argument of filter should be the data.frame. If numeric, will drop all observations with a matchscore lower than or equal to filter.
filter.args	list. Arguments passed to filter, if a function
evaluate	function to evalute merge_plus match
evaluate.args	list. Arguments passed to evaluate
chunk	character vector. Columns by which to group by for merging. All columns
match_type.	string. If 'exact', match is exact, if 'fuzzy', match is fuzzy.
amatch.args.	additional arguments for amatch, to be used if match_type = 'fuzzy'. Suggested defaults provided. (see amatch, method='jw')

Value

list with matches, filtered matches (if applicable), data1 and data2 minus matches, and match evaluation

See Also

match_evaluate

Examples

```
x <- data.frame('id' = 1:16, "a" = 10:25, "b" = 20:35, "c" = 30:45,
  "d" = rep(c("one", "two", "three", "four"), 4))
y <- data.frame('id' = 17:32, "a" = 10:25, "b" = c(20:30, 32:36), "c" = 30:45,
 "d" = rep(c("one", "two", "three", "four"), 4))
merge_plus_chunk(x, y, by = c('b'), unique_key_1 = 'id', unique_key_2 = 'id', chunk = c("a", "d"))
```

State_FIPS *State_FIPS*

Description

State FIPS codes

Usage

```
State_FIPS
```

Format

An object of class `data.frame` with 50 rows and 2 columns.

tier_match *tier_match*

Description

Constructs a tier_match by running merge_plus with different parameters sequentially on the same data, removing matched observations after each tier

Usage

```
tier_match(data1, data2, by = NULL, by.x = by, by.y = by,
  suffixes = c(".x", ".y"), check_merge = TRUE, unique_key_1, unique_key_2,
  tiers = list(), takeout = "both", match_type = "exact",
  amatch.args = list(method = "jw", p = 0.1, maxDist = 0.05, matchNA = FALSE),
  clean = clean_strings, clean.args = list(), score_settings = NULL,
  filter = NULL, filter.args = list(), evaluate = match_evaluate,
  evaluate.args = list())
```

Arguments

data1	data.frame. First to-merge dataset.
data2	data.frame. Second to-merge dataset.
suffixes	see merge
check_merge	logical. Checks that your unique_keys are indeed unique, and prevents merge from running if merge would result in data.frames larger than 5 million rows
unique_key_1	character vector. Primary key of data1 that uniquely identifies each row (can be multiple fields)
unique_key_2	character vector. Primary key of data2 that uniquely identifies each row (can be multiple fields)
tiers	list(). tier is a list of lists, where each list holds the parameters for creating that tier. All arguments to tier_match listed after this argument can either be supplied directly to tier_match, or indirectly via tiers.
takeout	string. Specifies whether to exclude matched observations from matching in subsequent tiers for 'data1', 'data2' or 'both'. of matching. If
clean	Function to clean strings prior to match. see clean_strings.
clean.args	list. Arguments passed to clean.
score_settings	list. score settings. See vingette matchscore
filter	function or numeric. Filters a merged data1-data2 dataset. If a function, should take in a data.frame (data1 and data2 merged by name1 and name2) and spit out a trimmed verion of the data.frame (fewer rows). Think of this function as applying other conditions to matches, other than a match by name. The first argument of filter should be the data.frame. If numeric, will drop all observations with a matchscore lower than or equal to filter.

filter.args	list. Arguments passed to filter, if a function
evaluate	Function to evalute merge_plus output. see evaluate_match.
evaluate.args	list. Arguments passed to function specified by evaluate
match_type.	string. If 'exact', match is exact, if 'fuzzy', match is fuzzy.
amatch.args.	additional arguments for amatch, to be used if match_type = 'fuzzy'. Suggested defaults provided. (see amatch, method='jw')

Value

list with matches, data1 and data2 minus matches, and match evaluation

See Also

merge_plus clean_strings

Examples

```
data1 = data.frame(unique_key = 1, name = c("hello world"))
data2 = data.frame(unique_key = 1:3, name = c("hello world", "Hello World", "hello worlds"))
tier_match(data1, data2, by='name', unique_key_1='unique_key', unique_key_2='unique_key',
        tiers = list(a=list(clean='none'), b=list(), c=list(match_type='fuzzy')), takeout='data2')
```

---

winsor *winsor*

Description

winsor

Usage

```
winsor(x, fraction = 0.05)
```

Arguments

x	numeric or vector. variable to be winsored
fraction	numeric. winsor cutoff

Value

x winsored at fraction

word_frequency *word_frequency*

Description

A function to count occurences in a set of strings; function does minimal cleaning (remove punctuation and extra spaces);

Usage

```
word_frequency(string)
```

Arguments

string	character vector

Value

data frame with word frequency

Examples

```
# with simple vector
names = c("company name 1", "company name 2", "company 3")
word_frequency(names)

# with a dataframe
col1 = c(1,2,3)
df1 = cbind(col1, names)
word_frequency(df1[,2])

# more complicated example
names = c("company name 1", "company &    3 inc.", "co nm 1,, lp")
df2 = cbind(col1, names)
word_frequency(df2[,2])
```

Index