Package ‘messy.cats’

November 30, 2022

Title Employs String Distance Tools to Help Clean Categorical Data

Version 1.0

BugReports https://github.com/hkarp1/messy.cats/issues

Description Matching with string distance has never been easier! 'messy.cats' contains various functions that employ string distance tools in order to make data management easier for users working with categorical data. Categorical data, especially user inputted categorical data that often tends to be plagued by typos, can be difficult to work with. 'messy.cats' aims to provide functions that make cleaning categorical data simple and easy.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.2.2

Depends R (>= 3.5.0)

Imports dplyr, stringdist, varhandle, rapportools, stringr, gt

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author Andrew Hennessy [aut],
Duncan Kreutter [aut],
Harrison Karp [cre]

Maintainer Harrison Karp <hkarp@wesleyan.edu>

Repository CRAN

Date/Publication 2022-11-30 11:40:02 UTC

R topics documented:

cat_join *cat_join*

Description

cat_join() joins two dataframes using the closest match between two specified columns with misspellings or slight format differences. The closest match can be found using a variety of different string distance measurement options.

Usage

```
cat_join(
  messy_df,
  clean_df,
  by,
  threshold = NA,
  method = "jw",
  q = 1,
  p = 0,
  bt = 0,
  useBytes = FALSE,
  weight = c(d = 1, i = 1, t = 1),
  join = "left"
)
```

Arguments

`messy_df` The dataframe to be joined using a messy categorical variable.

`clean_df` The dataframe to be joined with a clean categorical variable to be used as a reference for the messy column.

`by` A vector that specifies the columns to match and join by. If the column names are the same input: "column_name". If the columns have different names input: c("messy_column" = "clean_column")

`threshold` The maximum distance that will form a match. If this argument is specified, any element in the messy vector that has no match closer than the threshold distance will be replaced with NA. Default: NA

`method` The type of string distance calculation to use. Possible methods are : osa, lv, dl, hamming, lcs, qgram, cosine, jaccard, jw, and soundex. See package stringdist for more information. Default: 'jw'

`q` Size of the q-gram used in string distance calculation. Default: 1

`p` Only used with method "jw", the Jaro-Winkler penatly size. Default: 0

`bt` Only used with method "jw" with p > 0, Winkler's boost threshold. Default: 0

`useBytes` Whether or not to perform byte-wise comparison. Default: FALSE

`weight` Only used with methods "osa" or "dl", a vector representing the penalty for deletion, insertion, substitution, and transposition, in that order. Default: c(d = 1, i = 1, t = 1)

`join` Choose a join function from the dplyr package to use in joining the datasets. Default: 'left'

Details

When dealing with messy categorical string data, string distance matching can be an easy and efficient cleaning tool. A variety of string distance calculation algorithms have been developed for different types of data, and these algorithms can be used to detect and remedy problems with categorical string data.

By providing a correctly spelled and specified vector of categories to be compared against a vector of messy strings, a cleaned vector of categories can be generated by finding the correctly specificed string most similar to a messy string. This method works particularly well for messy user-inputted data that often suffers from transposition or misspelling errors.

`cat_join()` joins the messy and clean datasets using the closest matching elements from designated columns. The columns from the datasets are inputted into `cat_replace()` as the messy and clean vectors, and the datasets are joined using a user inputted dplyr join verb.

Value

Returns a dataframe consisting of the two inputted dataframes joined by their designated columns.

Examples

```
if(interactive()){
 #EXAMPLE1
 messy_trees = data.frame()
 messy_trees[1:9,1] = c("red oak", "williw", "hemluck", "white elm",
  "fir tree", "birch tree", "pone", "dagwood", "mople")
 messy_trees[1:9,2] = c(34,12,43,32,65,23,12,45,35)
 clean_trees=data.frame()
 clean_trees[1:9,1] = c("oak", "willow", "hemlock", "elm", "fir",
 "birch", "pine", "dogwood", "maple")
 clean_trees[1:9,2] = "y"
 cat_join(messy_trees,clean_trees,by="V1",method="jaccard")
 }
```

cat_match *cat_match*

Description

cat_match() matches the contents of a messy vector with the closest match in a clean vector. The closest match can be found using a variety of different string distance measurement options.

Usage

```
cat_match(
  messy_v,
  clean_v,
  return_dists = TRUE,
  return_lists = NA,
  pick_lists = FALSE,
  threshold = NA,
  method = "jw",
  q = 1,
  p = 0,
  bt = 0,
  useBytes = FALSE,
  weight = c(d = 1, i = 1, t = 1)
)
```

Arguments

messy_v The messy string vector that will be restructured. This can come in the form of a column of a dataframe or a lone vector.

clean_v The clean string vector that will be referenced to perform the restructing. Again, this argument can be a dataframe column or vector.

return_dists If set to TRUE the distance between the matched strings will be returned as a third column in the output dataframe, Default: TRUE `return_lists` Return list of top X matches, Default: NA

`pick_lists` Set to TRUE to manually choose matches, Default: F

`threshold` The maximum distance that will form a match. If this argument is specified, any element in the messy vector that has no match closer than the threshold distance will be replaced with NA. Default: NA

`method` The type of string distance calculation to use. Possible methods are : osa, lv, dl, hamming, lcs, qgram, cosine, jaccard, jw, and soundex. See package stringdist for more information. Default: ’jw’

`q` Size of the q-gram used in string distance calculation. Default: 1

`p` Only used with method "jw", the Jaro-Winkler penatly size. Default: 0

`bt` Only used with method "jw" with p > 0, Winkler’s boost threshold. Default: 0

`useBytes` Whether or not to perform byte-wise comparison. Default: FALSE

`weight` Only used with methods "osa" or "dl", a vector representing the penalty for deletion, insertion, substitution, and transposition, in that order. Default: c(d = 1, i = 1, t = 1)

Details

When dealing with messy categorical string data, string distance matching can be an easy and efficient cleaning tool. A variety of string distance calculation algorithms have been developed for different types of data, and these algorithms can be used to detect and remedy problems with categorical string data.

By providing a correctly spelled and specified vector of categories to be compared against a vector of messy strings, a cleaned vector of categories can be generated by finding the correctly specificed string most similar to a messy string. This method works particularly well for messy user-inputted data that often suffers from transposition or misspelling errors.

`cat_match()` is meant as an exploratory tool to discover how the elements of two vectors will match using string distance measures, and has added functionality to solve issues by hand and create a dataframe that can be used to create custom matches between the clean and messy vectors.

Value

Returns a dataframe with each unique value in the bad vector and it’s closest match in the good vector. If return_dists is TRUE the distances between the matches are added as a column.

Examples

```
if(interactive()){
 messy_trees = c("red oak", "williw", "hemluck", "white elm",
 "fir tree", "birch tree", "pone", "dagwood", "mople")
 clean_trees = c("oak", "willow", "hemlock", "elm", "fir", "birch", "pine", "dogwood", "maple")
 matched_trees = cat_match(messy_trees, clean_trees)
 }
```

cat_replace *cat_replace*

Description

cat_replace() replaces the contents of a messy vector with the closest match in a clean vector. The closest match can be found using a variety of different string distance measurement options.

Usage

```
cat_replace(
  messy_v,
  clean_v,
  threshold = NA,
  method = "jw",
  q = 1,
  p = 0,
  bt = 0,
  useBytes = FALSE,
  weight = c(d = 1, i = 1, t = 1)
)
```

Arguments

messy_v	The messy string vector that will be restructured. This can come in the form of a column of a dataframe or a lone vector.
clean_v	The clean string vector that will be referenced to perform the restructing. Again, this argument can be a dataframe column or vector.
threshold	The maximum distance that will form a match. If this argument is specified, any element in the messy vector that has no match closer than the threshold distance will be replaced with NA. Default: NA
method	The type of string distance calculation to use. Possible methods are : osa, lv, dl, hamming, lcs, qgram, cosine, jaccard, jw, and soundex. See package stringdist for more information. Default: 'jw'
q	Size of the q-gram used in string distance calculation. Default: 1
p	Only used with method "jw", the Jaro-Winkler penatly size. Default: 0
bt	Only used with method "jw" with p > 0, Winkler's boost threshold. Default: 0
useBytes	Whether or not to perform byte-wise comparison. Default: FALSE
weight	Only used with methods "osa" or "dl", a vector representing the penalty for deletion, insertion, substitution, and transposition, in that order. Default: c(d = 1, i = 1, t = 1)

Details

When dealing with messy categorical string data, string distance matching can be an easy and efficient cleaning tool. A variety of string distance calculation algorithms have been developed for different types of data, and these algorithms can be used to detect and remedy problems with categorical string data.

By providing a correctly spelled and specified vector of categories to be compared against a vector of messy strings, a cleaned vector of categories can be generated by finding the correctly specificed string most similar to a messy string. This method works particularly well for messy user-inputted data that often suffers from transposition or misspelling errors.

`cat_replace()` replaces the elements of the messy vector with the closest matching element from the clean vector.

Value

cat_replace() returns a cleaned version of the bad vector, with each element replaced by the most similar element of the good vector.

Examples

```
if(interactive()){
 messy_trees = c("red oak", "williw", "hemluck", "white elm", "fir tree",
  "birch tree", "pone", "dagwood", "mople")
 clean_trees = c("oak", "willow", "hemlock", "elm", "fir", "birch", "pine", "dogwood", "maple")
 cleaned_trees = cat_replace(messy_trees, clean_trees)

 }
```

`clean_caterpillars` *clean_caterpillars*

Description

Dataframe with caterpillar counts from three summers.

Usage

```
clean_caterpillars
```

Format

A data frame with 74 rows and 3 variables:

`species` character Full latin names of 29 caterpillar species.

`count` integer Randomly generated fake counts of the caterpillars.

`year` double Year of caterpillar observations.

Details

An example dataset with clean caterpillar species names.

`clean_names.df` *clean_names.df*

Description

Data set of clean names

Usage

```
clean_names.df
```

Format

A data frame with 20 rows and 2 variables:

`first` character Clean first names

`last` character Clean last names

Details

An example data that can be used in testing messy.cats functions

`country.names` *country.names*

Description

Dataframe with country names as only variable, contains many popular and official names for countries.

Usage

```
country.names
```

Format

A data frame with 203 rows and 1 variables:

`name` character Names of countries

Details

This dataframe contains a list of clean country names with many popular and official names for countries.

`country_match` *country_match*

Description

A wrapper function for `cat_match()` that only requires an inputted vector of messy country names. `country_match()` uses a built in clean list of country names `country.names` as the reference clean vector.

Usage

```
country_match(messy_countries, threshold = NA, p = 0)
```

Arguments

`messy_countries`	Vector containing the messy country names that will be replaced by the closest match from `country.names`
`threshold`	The maximum distance that will form a match. If this argument is specified, any element in the messy vector that has no match closer than the threshold distance will be replaced with NA. Default: NA
`p`	Only used with method "jw", the Jaro-Winkler penatly size. Default: 0

Details

Country names are often misspelled or abbreviated in datasets, especially datasets that have been manually digitized or created. `country_match()` is a warpper function of `cat_match()` that quickly solves this common issue of mispellings or different formats of country names across datasets. This wrapper function uses a built in clean list of country names `country.names` as the reference clean vector and matches your inputted messy vector of names to their nearest country in `country.names`.

Value

`country_match()` returns a cleaned version of the bad vector, with each element replaced by the most similar element of the good vector.

Examples

```
if(interactive()){
 #EXAMPLE1
 lst <- c("Conagoa", "Blearaus", "Venzesual", "Uruagsya", "England")
 matched <- country_match(lst)
 }
```

country_replace *country_replace*

Description

A wrapper function for `cat_replace()` that only requires an inputted vector of messy countries. `country_replace()` uses a built in clean list of country names `country.names` as the reference clean vector.

Usage

```
country_replace(messy_countries, threshold = NA, p = 0)
```

Arguments

`messy_countries`	Vector containing the messy country names that will be replaced by the closest match from `country.names`
`threshold`	The maximum distance that will form a match. If this argument is specified, any element in the messy vector that has no match closer than the threshold distance will be replaced with NA. Default: NA
`p`	Only used with method "jw", the Jaro-Winkler penatly size. Default: 0

Details

Country names are often misspelled or abbreviated in datasets, especially datasets that have been manually digitized or created. `country_replace()` is a warpper function of `cat_replace()` that quickly solves this common issue of mispellings or different formats of country names across datasets. This wrapper function uses a built in clean list of country names `country.names` as the reference clean vector and replaces your inputted messy vector of names to their nearest match in `country.names`.

Value

`country_replace()` returns a cleaned version of the bad vector, with each element replaced by the most similar element of the good vector.

Examples

```
if(interactive()){
 #EXAMPLE1
 lst <- c("Conagoa", "Blearaus", "Venzesual", "Uruagsya", "England")
 fixed <- country_replace(lst)
 }
```

fix_typos *fix_typos*

Description

This function is meant to allow users to fix typos in strings that are not normally found in dictionaries.

Usage

```
fix_typos(typo_v, thr, occ_ratio)
```

Arguments

`typo_v` vector of strings that will have its typos cleaned

`thr` the string distance maximum used to determine typos. This argument is specified as the percentage of a typo that should at most be expected to be insertions, additons, deletions, and transpositions.

`occ_ratio` the minimum ratio of correctly spelled words to their typo. This argument helps to weed out words that are similar but valid. For example commonly occurring valid names such as Adam and Amy will not be recognized as typos even though they are similar because they both appear often. Typos are recognized by their similarity in addition to their infrequent occurrence.

Details

There are great tools like the hunspell package that allow users to fix typos for words found in dictionaries, but these functions struggle to work for strings like proper nouns and other specific terminology not usually found in common dictionaries. This function uses the text being cleaned as a dictionary. It finds probable correctly spelled words based on their high occurrence and finds typos based on their low occurence. This is based on the theory that typos will appear as infrequently used words due no one using them on purpose, and they will be a short string distance from commonly occurring correctly spelled words.

Value

reformatted vector with typos replaced with correctly spelled words

Examples

```
if(interactive()){
 #EXAMPLE1
 }
```

fuzzy_rbind *fuzzy_rbind*

Description

fuzzy_rbind() binds dataframes based on columns with slightly different names.

Usage

```
fuzzy_rbind(
  df1,
  df2,
  threshold,
  method = "jw",
  q = 1,
  p = 0,
  bt = 0,
  useBytes = FALSE,
  weight = c(d = 1, i = 1, t = 1)
)
```

Arguments

df1	The first dataframe to be bound.
df2	The second dataframe to be bound.
threshold	The maximum string distance between column names, if the distance between columns is greater than this threshold the columns will not be bound.
method	The type of string distance calculation to use. Possible methods are : osa, lv, dl, hamming, lcs, qgram, cosine, jaccard, jw, and soundex. See package stringdist for more information. Default: 'jw', Default: 'jw'
q	Size of the q-gram used in string distance calculation. Default: 1
p	Only used with method "jw", the Jaro-Winkler penatly size. Default: 0
bt	Only used with method "jw" with p > 0, Winkler's boost threshold. Default: 0
useBytes	Whether or not to perform byte-wise comparison. Default: FALSE
weight	Only used with methods "osa" or "dl", a vector representing the penalty for deletion, insertion, substitution, and transposition, in that order. Default: c(d = 1, i = 1, t = 1)

Details

When using datasets often times column names are slightly different, and `fuzzy_rbind()` helps to bind dataframes using fuzzy matching of the column names.

Value

fuzzy_rbind() returns a dataframe that has bound the two inputted dataframes based on the closest matching columns, column names from dataframe 1 are preserved.

Examples

```
if(interactive()){
 mtcars_colnames_messy = mtcars
 colnames(mtcars_colnames_messy)[1:5] = paste0(colnames(mtcars)[1:5], "_17")
 colnames(mtcars_colnames_messy)[6:11] = paste0(colnames(mtcars)[6:11], "_2017")
 x = fuzzy_rbind(mtcars, mtcars_colnames_messy, .5)
 x = fuzzy_rbind(mtcars, mtcars_colnames_messy, .2)
 }
```

`messy_caterpillars` *messy_caterpillars*

Description

DATASET_DESCRIPTION

Usage

```
messy_caterpillars
```

Format

A data frame with 39 rows and 3 variables:

`CaterpillarSpecies` character Full latin names of 39 caterpillar species with spelling and formatting errors.

`Avg Weight (mg)` double Randomly generated fake weight data for each caterpillar species.

`Avg Length (cm)` double Randomly generated fake length data for each caterpillar species.

Details

An example dataset with messy caterpillar species names.

`messy_names.df` *messy_names.df*

Description

Data set of messy names

Usage

```
messy_names.df
```

Format

A data frame with 80 rows and 2 variables:

`first` character Messy first names

`last` character MEssy last names

Details

An example data set of messy names that can be used in testing messy.cats functions.

`messy_states1` *messy_states1*

Description

US State names with 1 character randomly changed.

Usage

```
messy_states1
```

Format

A data frame with 50 rows and 1 variables:

`messy_states1` character All 50 US states with 1 randomly changed character.

Details

An example dataset with mispelled US state names. The names have had 1 character randomly changed.

`messy_states2` *messy_states2*

Description

US State names with 2 characters randomly changed.

Usage

```
messy_states2
```

Format

A data frame with 50 rows and 1 variables:

`messy_states2` character All 50 US states with 2 randomly changed characters.

Details

An example dataset with mispelled US state names. The names have had 2 characters randomly changed.

`picked_list` *picked_list*

Description

Handpicked matches from the datasets in intro.rmd.

Usage

```
picked_list
```

Format

A data frame with 15 rows and 3 variables:

`bad` character column of bad car names

`match` character column of good car names

`dists` double string distance between the good and bad car names

Details

An example dataset of matched car names.

select_metric *select_metric*

Description

Uses heuristic algorithm to suggest a stringdist metric from among hamming, lv, osa, dl, lcs, jw

Usage

```
select_metric(messy_v, clean_v)
```

Arguments

messy_v	a messy vector of strings
clean_v	a vector of strings for messy_v to be matched against

Details

for each metric, measures certainty via the difference between the best matches for each word and the average of all matches for each word

Value

a string representing the suggested stringdist metric

See Also

stringdist

Examples

```
select_metric(c("aapple", "bamana", "clemtidne"), c("apple", "banana", "clementine"))
```

state.name *state.name*

Description

Testing data set of state names

Usage

```
state.name
```

Format

A data frame with 50 rows and 1 variables:

`states` character State names

Details

Testing data set of state names

`state_match` *state_match*

Description

A wrapper function for `cat_match()`hat only requires an inputted vector of messy states. `state_match()` uses a built in clean list of state names `state.name` as the reference clean vector.

Usage

```
state_match(messy_states, threshold = NA, p = 0)
```

Arguments

`messy_states`	Vector containing the messy state names that will be replaced by the closest match from `state.name`
`threshold`	The maximum distance that will form a match. If this argument is specified, any element in the messy vector that has no match closer than the threshold distance will be replaced with NA. Default: NA
`p`	Only used with method "jw", the Jaro-Winkler penatly size. Default: 0

Details

State names are often misspelled or abbreviated in datasets, especially datasets that have been manually digitized or created. `state_match()` is a warpper function of `cat_match()` that quickly solves this common issue of mispellings or different formats of country names across datasets. This wrapper function uses a built in clean list of country names `state.name` as the reference clean vector and matches your inputted messy vector of names to their nearest state in `state.name`.

Value

`state_match()` returns a cleaned version of the bad vector, with each element replaced by the most similar element of the good vector.

Examples

```
if(interactive()){
 #EXAMPLE1
 lst <- c("Indianaa", "Wisvconsin", "aLaska", "NewJersey", "Claifoarni")
 matched <- state_match(lst)
 }
```

state_replace *state_replace*

Description

A wrapper function for `cat_replace()` that only requires an inputted vector of messy US state names. `state_replace()` uses the built-in character vector `state.name` as the reference clean vector.

Usage

```
state_replace(messy_states, threshold = NA, p = 0)
```

Arguments

`messy_states`	Vector containing the messy state names that will be replaced by the closest match from `state.name`
`threshold`	The maximum distance that will form a match. If this argument is specified, any element in the messy vector that has no match closer than the threshold distance will be replaced with NA. Default: NA
`p`	Only used with method "jw", the Jaro-Winkler penatly size. Default: 0

Details

State names are often misspelled or abbreviated in datasets, especially datasets that have been manually digitized or created. `state_replace()` is a warpper function of `cat_replace()` that quickly solves this common issue of mispellings or different formats of state names across datasets. This wrapper function uses a built in clean list of country names `state.name` as the reference clean vector and replaces your inputted messy vector of names to their nearest match in `state.name`.

Value

`state_replace()` returns a cleaned version of the bad vector, with each element replaced by the most similar element of the good vector.

Examples

```
if(interactive()){
 #EXAMPLE1
 lst <- c("Indianaa", "Wisvconsin", "aLaska", "NewJersey", "Claifoarni")
 fixed <- state_replace(lst)
 }
```

`typos` *typos*

Description

Data set of words, some correctly spelled, some typos, with their occurrence in text

Usage

```
typos
```

Format

A data frame with 27 rows and 2 variables:

`occurrence` double number of times word appears in text

`species` character words in text

Details

An example data set that can be used in testing fix_typos().

Index