Package ‘fuzzyjoin’

May 15, 2020

Type Package

Title Join Tables Together on Inexact Matching

Version 0.1.6

Maintainer David Robinson <admiral.david@gmail.com>

Description Join tables together based not on whether columns match exactly, but whether they are similar by some comparison. Implementations include string distance and regular expression matching.

License MIT + file LICENSE

Encoding UTF-8

LazyData TRUE

VignetteBuilder knitr

Depends R (>= 2.10)

Imports stringdist, stringr, dplyr (>= 0.8.1), tidyr (>= 0.4.0), purrr, geosphere, tibble

Suggests testthat, knitr, ggplot2, qdapDictionaries, readr, rvest, rmarkdown, maps, IRanges, covr

RoxygenNote 7.1.0

URL https://github.com/dgrtwo/fuzzyjoin

BugReports https://github.com/dgrtwo/fuzzyjoin/issues

NeedsCompilation no

Author David Robinson [aut, cre], Jennifer Bryan [ctb], Joran Elias [ctb]

Repository CRAN

Date/Publication 2020-05-15 05:50:21 UTC
\textbf{R topics documented:}

\begin{itemize}
  \item difference_join ................................. 2
  \item distance_join ................................. 3
  \item fuzzy_join .................................. 5
  \item genome_join ................................. 7
  \item geo_join ................................... 8
  \item interval_join ............................... 11
  \item misspellings .................................. 12
  \item regex_join .................................. 13
  \item stringdist_join ............................. 15
\end{itemize}

\textbf{Index} \hspace{1cm} 17

\begin{itemize}
  \item difference_join
    \textit{Join two tables based on absolute difference between their columns}
\end{itemize}

\textbf{Description}

Join two tables based on absolute difference between their columns

\textbf{Usage}

\begin{verbatim}
difference_join(
  x,
  y,
  by = NULL,
  max_dist = 1,
  mode = "inner",
  distance_col = NULL
)

difference_inner_join(x, y, by = NULL, max_dist = 1, distance_col = NULL)

difference_left_join(x, y, by = NULL, max_dist = 1, distance_col = NULL)

difference_right_join(x, y, by = NULL, max_dist = 1, distance_col = NULL)

difference_full_join(x, y, by = NULL, max_dist = 1, distance_col = NULL)

difference_semi_join(x, y, by = NULL, max_dist = 1, distance_col = NULL)

difference_anti_join(x, y, by = NULL, max_dist = 1, distance_col = NULL)
\end{verbatim}

\textbf{Arguments}

\begin{itemize}
  \item \texttt{x} \hspace{1cm} A tbl
  \item \texttt{y} \hspace{1cm} A tbl
\end{itemize}
**distance_join**

Join two tables based on a distance metric of one or more columns

**Description**

This differs from `difference_join` in that it considers all of the columns together when computing distance. This allows it to use metrics such as Euclidean or Manhattan that depend on multiple columns. Note that if you are computing with longitude or latitude, you probably want to use `geo_join`.

**Usage**

```r
distance_join(
  x,
  y,
  by = NULL,
  max_dist = 1,
  method = c("euclidean", "manhattan"),
  mode = "inner",
  distance_col = NULL
)

distance_inner_join(
  x,
  y,
  by = NULL,
  method = "euclidean",
  max_dist = 1,
  distance_col = NULL
)
```

**Examples**

```r
library(dplyr)

head(iris)

sepal_lengths <- data_frame(Sepal.Length = c(5, 6, 7), Type = 1:3)

iris %>%
difference_inner_join(sepal_lengths, max_dist = .5)
```
distance_left_join(
  x,
  y,
  by = NULL,
  method = "euclidean",
  max_dist = 1,
  distance_col = NULL
)

distance_right_join(
  x,
  y,
  by = NULL,
  method = "euclidean",
  max_dist = 1,
  distance_col = NULL
)

distance_full_join(
  x,
  y,
  by = NULL,
  method = "euclidean",
  max_dist = 1,
  distance_col = NULL
)

distance_semi_join(
  x,
  y,
  by = NULL,
  method = "euclidean",
  max_dist = 1,
  distance_col = NULL
)

distance_anti_join(
  x,
  y,
  by = NULL,
  method = "euclidean",
  max_dist = 1,
  distance_col = NULL
)
fuzzy_join

Arguments

x A tbl
y A tbl
by Columns by which to join the two tables
max_dist Maximum distance to use for joining
method Method to use for computing distance, either euclidean (default) or manhattan.
mode One of "inner", "left", "right", "full" "semi", or "anti"
distance_col If given, will add a column with this name containing the distance between the two

Examples

library(dplyr)

head(iris)
sepal_lengths <- data_frame(Sepal.Length = c(5, 6, 7),
                          Sepal.Width = 1:3)

iris %>%
distance_inner_join(sepal_lengths, max_dist = 2)

fuzzy_join Join two tables based not on exact matches, but with a function describing whether two vectors are matched or not

Description

The match_fun argument is called once on a vector with all pairs of unique comparisons: thus, it should be efficient and vectorized.

Usage

fuzzy_join(
  x,
  y,
  by = NULL,
  match_fun = NULL,
  multi_by = NULL,
  multi_match_fun = NULL,
  index_match_fun = NULL,
  mode = "inner",
  ...
)

fuzzy_join(x, y, by = NULL, match_fun, ...)

Arguments

x A tbl
y A tbl
by Columns of each to join
match_fun Vectorized function given two columns, returning TRUE or FALSE as to whether
they are a match. Can be a list of functions one for each pair of columns specified in by (if a named list, it uses the names in x). If only one function is given it is used on all column pairs.
multi_by Columns to join, where all columns will be used to test matches together
multi_match_fun Function to use for testing matches, performed on all columns in each data frame simultaneously
index_match_fun Function to use for matching tables. Unlike match_fun and index_match_fun, this is performed on the original columns and returns pairs of indices.
mode One of "inner", "left", "right", "full" "semi", or "anti"
... Extra arguments passed to match_fun

Details

match_fun should return either a logical vector, or a data frame where the first column is logical. If the latter, the additional columns will be appended to the output. For example, these additional columns could contain the distance metrics that one is filtering on.

Note that as of now, you cannot give both match_fun and multi_match_fun- you can either compare each column individually or compare all of them.

Like in dplyr's join operations, fuzzy_join ignores groups, but preserves the grouping of x in the output.
**Description**

This is an extension of `interval_join` specific to genomic intervals. Genomic intervals include both a chromosome ID and an interval: items are only considered matching if the chromosome ID matches and the interval overlaps. Note that there must be three arguments to `by`, and that they must be in the order c("chromosome", "start", "end").

**Usage**

- `genome_join(x, y, by = NULL, mode = "inner", ...)`
- `genome_inner_join(x, y, by = NULL, ...)`
- `genome_left_join(x, y, by = NULL, ...)`
- `genome_right_join(x, y, by = NULL, ...)`
- `genome_full_join(x, y, by = NULL, ...)`
- `genome_semi_join(x, y, by = NULL, ...)`
- `genome_anti_join(x, y, by = NULL, ...)`

**Arguments**

- `x`: A tbl
- `y`: A tbl
- `by`: Names of columns to join on, in order c("chromosome", "start", "end"). A match will be counted only if the chromosomes are equal and the start/end pairs overlap.
- `mode`: One of "inner", "left", "right", "full", "semi", or "anti"
- `...`: Extra arguments passed on to `findOverlaps`

**Details**

All the extra arguments to `interval_join`, which are passed on to `findOverlaps`, work for `genome_join` as well. These include `maxgap` and `minoverlap`.

**Examples**

```r
library(dplyr)
```
geo_join

Join two tables based on a geo distance of longitudes and latitudes

Description

This allows joining based on combinations of longitudes and latitudes. If you are using a distance metric that is "not" based on latitude and longitude, use distance_join instead. Distances are calculated based on the distHaversine, distGeo, distCosine, etc methods in the geosphere package.

Usage

geo_join(
  x,
  y,
  by = NULL,
  max_dist,
  method = c("haversine", "geo", "cosine", "meeus", "vincentysphere",
             "vincentyellipsoid"),
  unit = c("miles", "km"),
  mode = "inner",
  distance_col = NULL,
  ...
)

x1 <- tibble(id1 = 1:4,
              chromosome = c("chr1", "chr1", "chr2", "chr2"),
              start = c(100, 200, 300, 400),
              end = c(150, 250, 350, 450))

x2 <- tibble(id2 = 1:4,
              chromosome = c("chr1", "chr2", "chr2", "chr1"),
              start = c(140, 210, 400, 300),
              end = c(160, 240, 415, 320))

if (requireNamespace("IRanges", quietly = TRUE)) {
  # note that the the third and fourth items don't join (even though
  # 300-350 and 300-320 overlap) since the chromosomes are different:
  genome_inner_join(x1, x2, by = c("chromosome", "start", "end"))

  # other functions:
  genome_full_join(x1, x2, by = c("chromosome", "start", "end"))
  genome_left_join(x1, x2, by = c("chromosome", "start", "end"))
  genome_right_join(x1, x2, by = c("chromosome", "start", "end"))
  genome_semi_join(x1, x2, by = c("chromosome", "start", "end"))
  genome_anti_join(x1, x2, by = c("chromosome", "start", "end"))
}
geo_join

geo_inner_join(
  x,
  y,
  by = NULL,
  method = "haversine",
  max_dist = 1,
  distance_col = NULL,
  ...
)

geo_left_join(
  x,
  y,
  by = NULL,
  method = "haversine",
  max_dist = 1,
  distance_col = NULL,
  ...
)

geo_right_join(
  x,
  y,
  by = NULL,
  method = "haversine",
  max_dist = 1,
  distance_col = NULL,
  ...
)

geo_full_join(
  x,
  y,
  by = NULL,
  method = "haversine",
  max_dist = 1,
  distance_col = NULL,
  ...
)

geo_semi_join(
  x,
  y,
  by = NULL,
  method = "haversine",
  max_dist = 1,
  distance_col = NULL,
  ...
)
geo_join

)

geo_anti_join(
  x,
  y,
  by = NULL,
  method = "haversine",
  max_dist = 1,
  distance_col = NULL,
  ...
)

Arguments

x      A tbl
y      A tbl
by     Columns by which to join the two tables
max_dist Maximum distance to use for joining
method  Method to use for computing distance: one of "haversine" (default), "geo", "cosine", "meeus", "vincentysphere", "vincentyellipsoid"
unit    Unit of distance for threshold (default "miles")
mode    One of "inner", "left", "right", "full", "semi", or "anti"
distance_col If given, will add a column with this name containing the geographical distance between the two
...     Extra arguments passed on to the distance method

Details

"Haversine" was chosen as default since in some tests it is approximately the fastest method. Note that by far the slowest method is vincentyellipsoid, and on fuzzy joins should only be used when there are very few pairs and accuracy is imperative.

If you need to use a custom geo method, you may want to write it directly with the multi_by and multi_match_fun arguments to fuzzy_join.

Examples

library(dplyr)
data("state")

# find pairs of US states whose centers are within
# 200 miles of each other
states <- data_frame(state = state.name,
  longitude = state.center$x,
  latitude = state.center$y)

s1 <- rename(states, state1 = state)
\begin{verbatim}

s2 <- rename(states, state2 = state)
pairs <- s1 %>%
    geo_inner_join(s2, max_dist = 200) %>%
    filter(state1 != state2)
pairs

# plot them
library(ggplot2)
ggplot(pairs, aes(x = longitude.x, y = latitude.x,
    xend = longitude.y, yend = latitude.y)) +
    geom_segment(color = "red") +
    borders("state") +
    theme_void()

# also get distances
s1 %>%
    geo_inner_join(s2, max_dist = 200, distance_col = "distance")

\end{verbatim}

\textbf{interval_join} \hspace{1cm} \textit{Join two tables based on overlapping (low, high) intervals}

\textbf{Description}

Joins tables based on overlapping intervals: for example, joining the row (1, 4) with (3, 6), but not with (5, 10). This operation is sped up using interval trees as implemented in the IRanges package. You can specify particular relationships between intervals (such as a maximum gap, or a minimum overlap) through arguments passed on to \texttt{findOverlaps}. See that documentation for descriptions of such arguments.

\textbf{Usage}

\begin{verbatim}

interval_join(x, y, by, mode = "inner", ...)
interval_inner_join(x, y, by = NULL, ...)
interval_left_join(x, y, by = NULL, ...)
interval_right_join(x, y, by = NULL, ...)
interval_full_join(x, y, by = NULL, ...)
interval_semi_join(x, y, by = NULL, ...)
interval_anti_join(x, y, by = NULL, ...)

\end{verbatim}
misspellings

A corpus of common misspellings, for examples and practice

Description

This is a codetbl_df mapping misspellings of their words, compiled by Wikipedia, where it is licensed under the CC-BY SA license. (Three words with non-ASCII characters were filtered out). If you’d like to reproduce this dataset from Wikipedia, see the example code below.

Arguments

- **x**: A tbl
- **y**: A tbl
- **by**: Columns by which to join the two tables. If provided, this must be two columns: start of interval, then end of interval
- **mode**: One of "inner", "left", "right", "full" "semi", or "anti"
- **...**: Extra arguments passed on to `findOverlaps`

Details

This allows joining on date or datetime intervals. It throws an error if the type of date/datetime disagrees between the two tables.

This requires the IRanges package from Bioconductor. See here for installation: https://bioconductor.org/packages/release/bioc/html/IRanges.html.

Examples

```r
if (requireNamespace("IRanges", quietly = TRUE)) {
  x1 <- data.frame(id1 = 1:3, start = c(1, 5, 10), end = c(3, 7, 15))
  x2 <- data.frame(id2 = 1:3, start = c(2, 4, 16), end = c(4, 8, 20))
  interval_inner_join(x1, x2)

  # Allow them to be separated by a gap with a maximum:
  interval_inner_join(x1, x2, maxgap = 1)  # let 1 join with 2
  interval_inner_join(x1, x2, maxgap = 20)  # everything joins each other

  # Require that they overlap by more than a particular amount
  interval_inner_join(x1, x2, minoverlap = 3)

  # other types of joins:
  interval_full_join(x1, x2)
  interval_left_join(x1, x2)
  interval_right_join(x1, x2)
  interval_semi_join(x1, x2)
  interval_anti_join(x1, x2)
}
```
Usage

misspellings

Format

An object of class tbl_df (inherits from tbl.data.frame) with 4505 rows and 2 columns.

Source


Examples

```r
## Not run:
library(rvest)
library(readr)
library(dplyr)
library(stringr)
library(tidyr)

h <- read_html(u)

misspellings <- h %>%
  html_nodes("pre") %>%
  html_text() %>%
  readr::read_delim(col_names = c("misspelling", "correct"), delim = ">",
                   skip = 1) %>%
  mutate(misspelling = str_sub(misspelling, 1, -2)) %>%
  unnest(correct = str_split(correct, ", ")) %>%
  filter(Encoding(correct) != "UTF-8")

## End(Not run)
```

regex_join

Join two tables based on a regular expression in one column matching the other

Description

Join a table with a string column by a regular expression column in another table
Usage

regex_join(x, y, by = NULL, mode = "inner", ignore_case = FALSE)
regex_inner_join(x, y, by = NULL, ignore_case = FALSE)
regex_left_join(x, y, by = NULL, ignore_case = FALSE)
regex_right_join(x, y, by = NULL, ignore_case = FALSE)
regex_full_join(x, y, by = NULL, ignore_case = FALSE)
regex_semi_join(x, y, by = NULL, ignore_case = FALSE)
regex_anti_join(x, y, by = NULL, ignore_case = FALSE)

Arguments

x A tbl
y A tbl
by Columns by which to join the two tables
mode One of "inner", "left", "right", "full", "semi", or "anti"
ignore_case Whether to be case insensitive (default no)

See Also

str_detect

Examples

library(dplyr)
library(ggplot2)
data(diamonds)
diamonds <- tbl_df(diamonds)

d <- data_frame(regex_name = c("Idea", "mium", "Good"),
                   type = 1:3)

# When they are inner_joined, only Good<->Good matches
diamonds %>%
  inner_join(d, by = c(cut = "regex_name"))

# but we can regex match them
diamonds %>%
  regex_inner_join(d, by = c(cut = "regex_name"))
stringdist_join

Join two tables based on fuzzy string matching of their columns

Description
Join two tables based on fuzzy string matching of their columns. This is useful, for example, in matching free-form inputs in a survey or online form, where it can catch misspellings and small personal changes.

Usage

```r
stringdist_join(
  x,
  y,
  by = NULL,
  max_dist = 2,
  method = c("osa", "lv", "dl", "hamming", "lcs", "qgram", "cosine", "jaccard", "jw", "soundex"),
  mode = "inner",
  ignore_case = FALSE,
  distance_col = NULL,
  ...
)
```

```r
stringdist_inner_join(x, y, by = NULL, distance_col = NULL, ...)
```

```r
stringdist_left_join(x, y, by = NULL, distance_col = NULL, ...)
```

```r
stringdist_right_join(x, y, by = NULL, distance_col = NULL, ...)
```

```r
stringdist_full_join(x, y, by = NULL, distance_col = NULL, ...)
```

```r
stringdist_semi_join(x, y, by = NULL, distance_col = NULL, ...)
```

```r
stringdist_anti_join(x, y, by = NULL, distance_col = NULL, ...)
```

Arguments

- `x`: A tbl
- `y`: A tbl
- `by`: Columns by which to join the two tables
- `max_dist`: Maximum distance to use for joining
- `method`: Method for computing string distance, see `stringdist-metrics` in the `stringdist` package.
- `mode`: One of "inner", "left", "right", "full", "semi", or "anti"
ignore_case  Whether to be case insensitive (default yes)
distance_col If given, will add a column with this name containing the difference between the two
...  Arguments passed on to stringdist

Details

If method = "soundex", the max_dist is automatically set to 0.5, since soundex returns either a 0 (match) or a 1 (no match).

Examples

```r
library(dplyr)
library(ggplot2)
data(diamonds)

d <- data_frame(approximate_name = c("Idea", "Premiums", "Premioom", 
  "VeryGood", "VeryGood", "Fair"),
  type = 1:6)

# no matches when they are inner-joined:
diamonds %>%
  inner_join(d, by = c(cut = "approximate_name"))

# but we can match when they're fuzzy joined
diamonds %>%
  stringdist_inner_join(d, by = c(cut = "approximate_name"))
```
Index

*Topic datasets
  misspellings, 12

difference_anti_join (difference_join), 2
difference_full_join (difference_join), 2
difference_inner_join (difference_join), 2
difference_left_join (difference_join), 2
difference_right_join (difference_join), 2
difference_semi_join (difference_join), 2
distance_anti_join (distance_join), 3
distance_full_join (distance_join), 3
distance_inner_join (distance_join), 3
distance_left_join (distance_join), 3
distance_right_join (distance_join), 3
distance_semi_join (distance_join), 3
findOverlaps, 7, 11, 12
fuzzy_anti_join (fuzzy_join), 5
fuzzy_full_join (fuzzy_join), 5
fuzzy_inner_join (fuzzy_join), 5
fuzzy_join, 5
fuzzy_left_join (fuzzy_join), 5
fuzzy_right_join (fuzzy_join), 5
fuzzy_semi_join (fuzzy_join), 5

geo_anti_join (geo_join), 8
geno_full_join (geo_join), 8
geno_inner_join (geo_join), 8
geno_join, 8
geno_left join (geo_join), 8
geno_right join (geo_join), 8
geno_semi join (geo_join), 8
interval_anti_join (interval_join), 11
interval_full_join (interval_join), 11
interval_inner_join (interval_join), 11
interval_join, 7, 11
interval_left_join (interval_join), 11
interval_right_join (interval_join), 11
interval_semi_join (interval_join), 11
misspellings, 12

regex_anti_join (regex_join), 13
regex_full_join (regex_join), 13
regex_inner_join (regex_join), 13
regex_join, 13
regex_left_join (regex_join), 13
regex_right_join (regex_join), 13
regex_semi_join (regex_join), 13

str_detect, 14
stringdist, 16
stringdist_anti_join (stringdist_join), 15
stringdist_full_join (stringdist_join), 15
stringdist_inner_join (stringdist_join), 15
stringdist_join, 15
stringdist_left_join (stringdist_join), 15
stringdist_right_join (stringdist_join), 15
stringdist_semi_join (stringdist_join), 15