Package ‘widyr’

April 12, 2020

Type Package

Title Widen, Process, then Re-Tidy Data

Version 0.1.3

Description Encapsulates the pattern of untidying data into a wide matrix, performing some processing, then turning it back into a tidy form. This is useful for several operations such as co-occurrence counts, correlations, or clustering that are mathematically convenient on wide matrices.

License MIT + file LICENSE

Encoding UTF-8

LazyData TRUE

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

URL http://github.com/dgrtwo/widyr

BugReports http://github.com/dgrtwo/widyr/issues

VignetteBuilder knitr

Imports rlang, dplyr, tidyr, reshape2, tidytext, purrr, Matrix, broom

Suggests ggraph, igraph, gapminder, testthat, covr, knitr, janeaustenr, rmarkdown, unrvotes (>= 0.2.0), countrycode, fuzzyjoin, ggplot2, maps, iriba

RoxygenNote 7.1.0

NeedsCompilation no

Author David Robinson [aut, cre], Kanishka Misra [ctb], Julia Silge [ctb]

Repository CRAN

Date/Publication 2020-04-12 06:00:02 UTC
\textbf{\texttt{cor\_sparse} topics documented:}

- \texttt{cor\_sparse} .................................................. 2
- \texttt{pairwise\_cor} ........................................... 3
- \texttt{pairwise\_count} ......................................... 4
- \texttt{pairwise\_delta} .......................................... 5
- \texttt{pairwise\_dist} ............................................ 6
- \texttt{pairwise\_pmi} ............................................ 7
- \texttt{pairwise\_similarity} ................................. 8
- \texttt{squarely} ................................................ 9
- \texttt{widely} .................................................. 10
- \texttt{widely\_svd} ............................................ 12

\textbf{Index} 14

\begin{tabular}{ll}
\texttt{cor\_sparse} & \textit{Find the Pearson correlation of a sparse matrix efficiently} \\
\end{tabular}

\textbf{Description}

Find the Pearson correlation of a sparse matrix. For large sparse matrix this is more efficient in time and memory than \texttt{cor(as.matrix(x))}. Note that it does not currently work on \texttt{simple\_triplet\_matrix} objects.

\textbf{Usage}

\texttt{cor\_sparse(x)}

\textbf{Arguments}

- \texttt{x} A matrix, potentially a sparse matrix such as a "\texttt{dgTMatrix}" object

\textbf{Source}

This code comes from mike on this Stack Overflow answer: \url{http://stackoverflow.com/a/9626089/712603}.
pairwise_cor

---

**pairwise_cor** *Correlations of pairs of items*

**Description**

Find correlations of pairs of items in a column, based on a "feature" column that links them together. This is an example of the spread-operate-retidy pattern.

**Usage**

```r
pairwise_cor(
  tbl,
  item,
  feature,
  value,
  method = c("pearson", "kendall", "spearman"),
  use = "everything",
  ...
)
```

```r
pairwise_cor_(
  tbl,
  item,
  feature,
  value,
  method = c("pearson", "kendall", "spearman"),
  use = "everything",
  ...
)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tbl</code></td>
<td>Table</td>
</tr>
<tr>
<td><code>item</code></td>
<td>Item to compare; will end up in <code>item1</code> and <code>item2</code> columns</td>
</tr>
<tr>
<td><code>feature</code></td>
<td>Column describing the feature that links one item to others</td>
</tr>
<tr>
<td><code>value</code></td>
<td>Value column. If not given, defaults to all values being 1 (thus a binary correlation)</td>
</tr>
<tr>
<td><code>method</code></td>
<td>Correlation method</td>
</tr>
<tr>
<td><code>use</code></td>
<td>Character string specifying the behavior of correlations with missing values; passed on to <code>cor</code></td>
</tr>
<tr>
<td><code>...</code></td>
<td>Extra arguments passed on to <code>squarely</code>, such as <code>diag</code> and <code>upper</code></td>
</tr>
</tbody>
</table>
Examples

```r
library(dplyr)
library(gapminder)

gapminder %>%
  pairwise_cor(country, year, lifeExp)

gapminder %>%
  pairwise_cor(country, year, lifeExp, sort = TRUE)

# United Nations voting data
library(unvotes)

country_cors <- un_votes %>%
  mutate(vote = as.numeric(vote)) %>%
  pairwise_cor(country, rcid, vote, sort = TRUE)

country_cors
```

---

**pairwise_count**  
*Count pairs of items within a group*

**Description**

Count the number of times each pair of items appear together within a group defined by "feature." For example, this could count the number of times two words appear within documents.

**Usage**

```r
pairwise_count(tbl, item, feature, wt = NULL, ...)

pairwise_count_(tbl, item, feature, wt = NULL, ...)
```

**Arguments**

- `tbl`  
  Table

- `item`  
  Item to count pairs of; will end up in `item1` and `item2` columns

- `feature`  
  Column within which to count pairs `item2` columns

- `wt`  
  Optionally a weight column, which should have a consistent weight for each `feature`

  ...  
  Extra arguments passed on to `squarely`, such as `diag`, `upper`, and `sort`

**See Also**

`squarely`
Examples

```r
library(dplyr)
dat <- tibble(group = rep(1:5, each = 2),
              letter = c("a", "b",
                        "a", "c",
                        "a", "c",
                        "b", "e",
                        "b", "f"))

# count the number of times two letters appear together
pairwise_count(dat, letter, group)
pairwise_count(dat, letter, group, sort = TRUE)
pairwise_count(dat, letter, group, sort = TRUE, diag = TRUE)
```

---

pairwise_delta  

*Delta measure of pairs of documents*

Description

Compute the delta distances (from its two variants) of all pairs of documents in a tidy table.

Usage

```r
pairwise_delta(tbl, item, feature, value, method = "burrows", ...)
pairwise_delta_(tbl, item, feature, value, method = "burrows", ...)
```

Arguments

- `tbl`: Table
- `item`: Item to compare; will end up in `item1` and `item2` columns
- `feature`: Column describing the feature that links one item to others
- `value`: Value
- `method`: Distance measure to be used; see `dist`
- `...`: Extra arguments passed on to `squarely`, such as `diag` and `upper`

See Also

`squarely`
Examples

```r
library(janeaustenr)
library(dplyr)
library(tidytext)

# closest documents in terms of 1000 most frequent words
closest <- austen_books() %>%
  unnest_tokens(word, text) %>%
  count(book, word) %>%
  top_n(1000, n) %>%
  pairwise_delta(book, word, n, method = "burrows") %>%
  arrange(delta)

closest

closest %>%
  filter(item1 == "Pride & Prejudice")

# to remove duplicates, use upper = FALSE
closest <- austen_books() %>%
  unnest_tokens(word, text) %>%
  count(book, word) %>%
  top_n(1000, n) %>%
  pairwise_delta(book, word, n, method = "burrows", upper = FALSE) %>%
  arrange(delta)

# Can also use Argamon's Linear Delta
closest <- austen_books() %>%
  unnest_tokens(word, text) %>%
  count(book, word) %>%
  top_n(1000, n) %>%
  pairwise_delta(book, word, n, method = "argamon", upper = FALSE) %>%
  arrange(delta)
```

---

**pairwise_dist**

**Distances of pairs of items**

**Description**

Compute distances of all pairs of items in a tidy table.

**Usage**

```r
pairwise_dist(tbl, item, feature, value, method = "euclidean", ...)

pairwise_dist_(tbl, item, feature, value, method = "euclidean", ...)
```
pairwise_pmi

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tbl</td>
<td>Table</td>
</tr>
<tr>
<td>item</td>
<td>Item to compare; will end up in item1 and item2 columns</td>
</tr>
<tr>
<td>feature</td>
<td>Column describing the feature that links one item to others</td>
</tr>
<tr>
<td>value</td>
<td>Value</td>
</tr>
<tr>
<td>method</td>
<td>Distance measure to be used; see dist</td>
</tr>
<tr>
<td>...</td>
<td>Extra arguments passed on to squarely, such as diag and upper</td>
</tr>
</tbody>
</table>

See Also

squarely

Examples

library(gapminder)
library(dplyr)

# closest countries in terms of life expectancy over time
closest <- gapminder %>%
  pairwise_dist(country, year, lifeExp) %>%
  arrange(distance)

closest

closest %>%
  filter(item1 == "United States")

# to remove duplicates, use upper = FALSE
gapminder %>%
  pairwise_dist(country, year, lifeExp, upper = FALSE) %>%
  arrange(distance)

# Can also use Manhattan distance
gapminder %>%
  pairwise_dist(country, year, lifeExp, method = "manhattan", upper = FALSE) %>%
  arrange(distance)

pairwise_pmi  Pointwise mutual information of pairs of items

Description

Find pointwise mutual information of pairs of items in a column, based on a "feature" column that links them together. This is an example of the spread-operate-retain pattern.
Usage

pairwise_pmi(tbl, item, feature, sort = FALSE, ...)

pairwise_pmi_(tbl, item, feature, sort = FALSE, ...)

Arguments

tbl  Table
item  Item to compare; will end up in item1 and item2 columns
feature  Column describing the feature that links one item to others
sort  Whether to sort in descending order of the pointwise mutual information
...  Extra arguments passed on to squarely, such as diag and upper

Value

A tbl_df with three columns, item1, item2, and pmi.

Examples

library(dplyr)

dat <- tibble(group = rep(1:5, each = 2),
  letter = c("a", "b",
    "a", "c",
    "a", "c",
    "b", "e",
    "b", "f"))

# how informative is each letter about each other letter
pairwise_pmi(dat, letter, group)
pairwise_pmi(dat, letter, group, sort = TRUE)

pairwise_similarity  Cosine similarity of pairs of items

Description

Compute cosine similarity of all pairs of items in a tidy table.

Usage

pairwise_similarity(tbl, item, feature, value, ...)

pairwise_similarity_(tbl, item, feature, value, ...)
Arguments

- **tbl**: Table
- **item**: Item to compare; will end up in `item1` and `item2` columns
- **feature**: Column describing the feature that links one item to others
- **value**: Value
- **...**: Extra arguments passed on to `squarely`, such as `diag` and `upper`

See Also

- `squarely`

Examples

```r
library(janeaustenr)
library(dplyr)
library(tidytext)

# Comparing Jane Austen novels
austen_words <- austen_books() %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words, by = "word") %>%
  count(book, word) %>%
  ungroup()

# closest books to each other
closest <- austen_words %>%
  pairwise_similarity(book, word, n) %>%
  arrange(desc(similarity))

closest

closest %>%
  filter(item1 == "Emma")
```

**squarely**

*A special case of the widely adverb for creating tidy square matrices*

Description

A special case of `widely`. Used to pre-prepare and post-tidy functions that take an m x n (m items, n features) matrix and return an m x m (item x item) matrix, such as a distance or correlation matrix.

Usage

```r
squarely(.f, diag = FALSE, upper = TRUE, ...)

squarely_(.f, diag = FALSE, upper = TRUE, ...)
```
Arguments

.f Function to wrap
diag Whether to include diagonal (i = j) in output
upper Whether to include upper triangle, which may be duplicated
... Extra arguments passed on to widely

Value

Returns a function that takes at least four arguments:

tbl A table
item Name of column to use as rows in wide matrix
feature Name of column to use as columns in wide matrix
feature Name of column to use as values in wide matrix
... Arguments passed on to inner function

See Also

widely, pairwise_count, pairwise_cor, pairwise_dist

Examples

library(dplyr)
library(gapminder)

closest_continent <- gapminder %>%
  group_by(continent) %>%
  squarely(dist)(country, year, lifeExp)

widely  Adverb for functions that operate on matrices in "wide" format

Description

Modify a function in order to pre-cast the input into a wide matrix format, perform the function, and then re-tidy (e.g. melt) the output into a tidy table.

Usage

widely(.f, sort = FALSE, sparse = FALSE, maximum_size = 1e+07)
widely_(.f, sort = FALSE, sparse = FALSE, maximum_size = 1e+07)
**Arguments**

- `.f` Function being wrapped
- `sort` Whether to sort in descending order of value
- `sparse` Whether to cast to a sparse matrix
- `maximum_size` To prevent crashing, a maximum size of a non-sparse matrix to be created. Set to NULL to allow any size matrix.

**Value**

Returns a function that takes at least four arguments:

- `tbl` A table
- `row` Name of column to use as rows in wide matrix
- `column` Name of column to use as columns in wide matrix
- `value` Name of column to use as values in wide matrix
- `...` Arguments passed on to inner function

`widely` creates a function that takes those columns as bare names, `widely_` a function that takes them as strings.

**Examples**

```r
library(dplyr)
library(gapminder)

gapminder

gapminder %>%
  widely(dist)(country, year, lifeExp)

# can perform within groups
closest_continent <- gapminder %>%
  group_by(continent) %>%
  widely(dist)(country, year, lifeExp)
closest_continent

# for example, find the closest pair in each
closest_continent %>%
  top_n(1, -value)
```
widely_svd  

*Turn into a wide matrix, perform SVD, return to tidy form*

**Description**

This is useful for dimensionality reduction of items, especially when setting a lower `nv`.

**Usage**

```r
widely_svd(tbl, item, feature, value, nv = NULL, weight_d = FALSE, ...)
widely_svd_(tbl, item, feature, value, nv = NULL, weight_d = FALSE, ...)
```

**Arguments**

- `tbl` Table
- `item` Item to perform dimensionality reduction on; will end up in `item` column
- `feature` Column describing the feature that links one item to others.
- `value` Value
- `nv` Optional; the number of principal components to estimate. Recommended for matrices with many features.
- `weight_d` Whether to multiply each value by the `d` principal component.
- `...` Extra arguments passed to `svd` (if `nv` is NULL) or `irlba` (if `nv` is given)

**Value**

A `tbl_df` with three columns. The first is retained from the `item` input, then `dimension` and `value`. Each row represents one principal component value.

**Examples**

```r
library(dplyr)
library(gapminder)

# principal components driving change
gapminder_svd <- gapminder %>%
    widely_svd(country, year, lifeExp)

gapminder_svd

# compare SVDs, join with other data
library(ggplot2)
library(tidyr)

gapminder_svd %>%
    spread(dimension, value) %>%
```
inner_join(distinct(gapminder, country, continent), by = "country") %>%
ggplot(aes('1', '2', label = country)) +
geom_point(aes(color = continent)) +
geom_text(vjust = 1, hjust = 1)
Index

cor_sparse, 2
dist, 5, 7

pairwise_cor, 3, 10
pairwise_cor_ (pairwise_cor), 3
pairwise_count, 4, 10
pairwise_count_ (pairwise_count), 4
pairwise_delta, 5
pairwise_delta_ (pairwise_delta), 5
pairwise_dist, 6, 10
pairwise_dist_ (pairwise_dist), 6
pairwise_pmi, 7
pairwise_pmi_ (pairwise_pmi), 7
pairwise_similarity, 8
pairwise_similarity_
  (pairwise_similarity), 8

squarely, 4, 5, 7, 9, 9
squarely_ (squarely), 9

widely, 9, 10, 10
widely_ (widely), 10
widely_svd, 12
widely_svd_ (widely_svd), 12