Package ‘widyr’

August 12, 2021

Type Package
Title Widen, Process, then Re-Tidy Data
Version 0.1.4
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
Maintainer David Robinson <admiral.david@gmail.com>
URL https://github.com/dgrtwo/widyr
BugReports https://github.com/dgrtwo/widyr/issues
VignetteBuilder knitr
Imports rlang, dplyr, tidyr, reshape2, tidytext, purrr, Matrix, broom, tibble, gapminder
Suggests ggraph, igraph, testthat, covr, knitr, janeaustenr, rmarkdown, unvotes (>= 0.3.0), countrycode, fuzzyjoin, ggplot2, maps, irlba
RoxygenNote 7.1.1
NeedsCompilation no
Author David Robinson [aut, cre], Kanishka Misra [ctb], Julia Silge [ctb]
Repository CRAN
Date/Publication 2021-08-12 17:10:02 UTC

R topics documented:
cor_sparse ................................................................. 2
pairwise_cor ............................................................. 2
**pairwise_cor**

pairwise_count ................................................. 4
pairwise_delta ................................................. 5
pairwise_dist ..................................................... 6
pairwise_pmi ...................................................... 7
pairwise_similarity .............................................. 8
squarely ............................................................ 9
widely .............................................................. 10
widely_hclust ...................................................... 11
widely_kmeans ..................................................... 12
widely_svd ........................................................ 13

**Index**

15

---

### cor_sparse

Find the Pearson correlation of a sparse matrix efficiently

**Description**

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

**Usage**

```r
cor_sparse(x)
```

**Arguments**

- `x` A matrix, potentially a sparse matrix such as a "dgTMatrix" object

**Source**

This code comes from mike on this Stack Overflow answer: [https://stackoverflow.com/a/9626089/712603](https://stackoverflow.com/a/9626089/712603).

---

### 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.
pairwise_cor

Usage

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

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 column. If not given, defaults to all values being 1 (thus a binary correlation)
  method    Correlation method
  use       Character string specifying the behavior of correlations with missing values; passed on to cor
  ...       Extra arguments passed on to squarely, such as diag and upper

Examples

library(dplyr)
library(gapminder)

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

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

# United Nations voting data
if (require("unvotes", quietly = TRUE)) {
  country_cors <- un_votes %>%

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

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

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

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

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 count pairs of; will end up in item1 and item2 columns</td>
</tr>
<tr>
<td>feature</td>
<td>Column within which to count pairs item2 columns</td>
</tr>
<tr>
<td>wt</td>
<td>Optionally a weight column, which should have a consistent weight for each feature</td>
</tr>
<tr>
<td>...</td>
<td>Extra arguments passed on to squarely, such as diag, upper, and sort</td>
</tr>
</tbody>
</table>

See Also

squarely

Examples

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

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

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

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(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
pairwise_dist <- 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", ...)
```

**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(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-retidy pattern.

Usage

```r
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

```r
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

```r
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

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

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

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

```r
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.
widely_hclust

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_hclust  Cluster pairs of items into groups using hierarchical clustering

Description

Reshape a table that represents pairwise distances into hierarchical clusters, returning a table with **item** and **cluster** columns.

Usage

```r
widely_hclust(tbl, item1, item2, distance, k = NULL, h = NULL)
```
widely_kmeans

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tbl</td>
<td>Table</td>
</tr>
<tr>
<td>item1</td>
<td>First item</td>
</tr>
<tr>
<td>item2</td>
<td>Second item</td>
</tr>
<tr>
<td>distance</td>
<td>Distance column</td>
</tr>
<tr>
<td>k</td>
<td>The desired number of groups</td>
</tr>
<tr>
<td>h</td>
<td>Height at which to cut the hierarchically clustered tree</td>
</tr>
</tbody>
</table>

See Also

cutree

Examples

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

# Construct Euclidean distances between countries based on life expectancy over time
country_distances <- gapminder %>%
  pairwise_dist(country, year, lifeExp)
country_distances

# Turn this into 5 hierarchical clusters
clusters <- country_distances %>%
  widely_hclust(item1, item2, distance, k = 8)

# Examine a few such clusters
clusters %>% filter(cluster == 1)
clusters %>% filter(cluster == 2)
```

widely_kmeans  
Cluster items based on k-means across features

Description

Given a tidy table of features describing each item, perform k-means clustering using \texttt{kmeans} and retidy the data into one-row-per-cluster.

Usage

\texttt{widely_kmeans(tbl, item, feature, value, k, fill = 0, ...)}


Arguments

- tbl: Table
- item: Item to cluster (as a bare column name)
- feature: Feature column (dimension in clustering)
- value: Value column
- k: Number of clusters
- fill: What to fill in for missing values
- ...: Other arguments passed on to kmeans

See Also

widely_hclust

Examples

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

clusters <- gapminder %>%
    widely_kmeans(country, year, lifeExp, k = 5)

clusters

clusters %>%
    count(cluster)

# Examine a few clusters
clusters %>%
    filter(cluster == 1)
clusters %>%
    filter(cluster == 2)
```

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

cutree, 12

dist, 5, 6

kmeans, 12, 13

pairwise_cor, 2, 10
pairwise_cor_ (pairwise_cor), 2
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, 4–6, 8, 9
squarely_ (squarely), 9

widely, 9, 10, 10
widely_ (widely), 10
widely_hclust, 11, 13
widely_kmeans, 12
widely_svd, 13
widely_svd_ (widely_svd), 13