Package ‘corpus’

December 12, 2017

Version 0.10.0
Title Text Corpus Analysis
Depends R (>= 3.3),
Imports stats, utf8 (>= 1.1.0)
Suggests knitr, Matrix, testthat
Enhances quanteda, tm
Description Text corpus data analysis, with full support for international text (Unicode). Functions for reading data from newline-delimited 'JSON' files, for normalizing and tokenizing text, for searching for term occurrences, and for computing term occurrence frequencies, including n-grams.
License Apache License (== 2.0) | file LICENSE
BugReports https://github.com/patperry/r-corpus/issues
LazyData Yes
Encoding UTF-8
VignetteBuilder knitr
NeedsCompilation yes
Author Patrick O. Perry [aut, cph, cre],
Finn Årup Nielsen [cph, dtc] (AFINN Sentiment Lexicon),
Martin Porter and Richard Boulton [ctb, cph, dtc] (Snowball Stemmer and Stopword Lists),
The Regents of the University of California [ctb, cph] (Strtod Library Procedure),
Carlo Strapparava and Alessandro Valitutti [cph, dtc] (WordNet-Affect Lexicon),
Unicode, Inc. [cph, dtc] (Unicode Character Database)
Maintainer Patrick O. Perry <pperry@stern.nyu.edu>
Repository CRAN
Date/Publication 2017-12-12 22:10:07 UTC
The Corpus Package

Description

Text corpus analysis functions

Details

This package contains functions for text corpus analysis. To create a text object, use the `read_ndjson` or `as_corpus_text` function. To split text into sentences or token blocks, use `text_split`. To specify preprocessing behavior for transforming a text into a token sequence, use `text_filter`. To tokenize text or compute term frequencies, use `text_tokens`, `term_stats` or `term_matrix`. To search for or count specific terms, use `text_locate`, `text_count`, or `text_detect`.

For a complete list of functions, use `library(help = "corpus")`.

Author(s)

Patrick O. Perry
Description

Lists of common abbreviations.

Usage

abbreviations_de
abbreviations_en
abbreviations_es
abbreviations_fr
abbreviations_it
abbreviations_pt
abbreviations_ru

Format

A character vector of unique abbreviations.

Details

The abbreviations_ objects are character vectors of abbreviations. These are words or phrases containing full stops (periods, ambiguous sentence terminators) that require special handling for sentence detection and tokenization.

The original lists were compiled by the Unicode Common Locale Data Repository. We have tailored the English list by adding single-letter abbreviations and making a few other additions.

The built-in abbreviation lists are reasonable defaults, but they may require further tailoring to suit your particular task.

See Also

text_filter.

affect_wordnet   WordNet-Affect Lexicon
Description

The WordNet-Affect Lexicon is a hand-curate collection of emotion-related words (nouns, verbs, adjectives, and adverbs), classified as “Positive”, “Negative”, “Neutral”, or “Ambiguous” and categorized into 28 subcategories (“Joy”, “Love”, “Fear”, etc.).

Terms can and do appear in multiple categories.

The original lexicon contains multi-word phrases, but they are excluded here. Also, we removed the term ‘thing’ from the lexicon.

The original WordNet-Affect lexicon is distributed as part of the WordNet Domains project, which is licensed under a Creative Commons Attribution 3.0 Unported License. You are free to share and adapt the lexicon, as long as you give attribution to the original authors.

Usage

affect_wordnet

Format

A data frame with one row for each term classification.

Source

http://wndomains.fbk.eu/wnaffect.html

References


---

corpus_frame  Corpus Data Frame

description

Create or test for corpus objects.

Usage

corpus_frame(..., row.names = NULL, filter = NULL)

as_corpus_frame(x, filter = NULL, ..., row.names = NULL)

is_corpus_frame(x)
Arguments

... data frame columns for corpus_frame; further arguments passed to as_corpus_text
from as_corpus_frame.

row.names character vector of row names for the corpus object.

filter text filter object for the "text" column in the corpus object.

x object to be coerced or tested.

Details

These functions create or convert another object to a corpus object. A corpus object is just a data
frame with special functions for printing, and a column names "text" of type "corpus_text".
corpus has similar semantics to the data.frame function, except that string columns do not get
converted to factors.
as_corpus_frame converts another object to a corpus data frame object. By default, the method
converts x to a data frame with a column named "text" of type "corpus_text", and sets the class
attribute of the result to c("corpus_frame", "data.frame").
is_corpus_frame tests whether x is a data frame with a column named "text" of type "corpus_text".
as_corpus_frame is generic: you can write methods to handle specific classes of objects.

Value

corpus_frame creates a data frame with a column named "text" of type "corpus_text", and a
class attribute set to c("corpus_frame", "data.frame").
as_corpus_frame attempts to coerce its argument to a corpus data frame object, setting the row.names
and calling as_corpus_text on the "text" column with the filter and ... arguments.
is_corpus_frame returns TRUE or FALSE depending on whether its argument is a valid corpus
object or not.

See Also

corpus-package, print.corpus_frame, corpus_text, read_ndjson.

Examples

# convert a data frame:
emoji <- data.frame(text = sapply(0x1f600 + 1:30, intToUtf8),
                  stringsAsFactors = FALSE)
as_corpus_frame(emoji)

# construct directly (no need for stringsAsFactors = FALSE):
corpus_frame(text = sapply(0x1f600 + 1:30, intToUtf8))

# convert a character vector:
as_corpus_frame(c(a = "goodnight", b = "moon")) # keeps names
as_corpus_frame(c(a = "goodnight", b = "moon"), row.names = NULL) # drops names
**Description**

Create or test for text objects.

**Usage**

```r
as_corpus_text(x, filter = NULL, ..., names = NULL)
```

```r
is_corpus_text(x)
```

**Arguments**

- `x` object to be coerced or tested.
- `filter` if non-NULL, a text filter for the converted result.
- `...` text filter properties to set on the result.
- `names` if non-NULL character vector of names for the converted result.

**Details**

The `corpus_text` type is a new data type provided by the corpus package suitable for processing international (Unicode) text. Text vectors behave like character vectors (and can be converted to them with the `as.character` function). They can be created using the `read_ndjson` function or by converting another object using the `as_corpus_text` function.

All text objects have a `text_filter` property specify how to transform the text into tokens or segment it into sentences.

The default behavior for `as_corpus_text` is to proceed as follows:

1. If `x` is a character vector, then we create a new text vector from `x`.
2. If `x` is a data frame, then we call `as_corpus_text` on `x$text` if a column named "text" exists in the data frame. If the data frame does not have a column named "text", then we fail with an error message.
3. If `x` is a `corpus_text` object, then we drop all attributes and we set the class to "corpus_text".
4. The default behavior for when none of the above conditions are true is to call `as.character` on the object first, preserving the names, and then and call `as_corpus_text` on the returned character object.

In all cases, when the names is `NULL`, we set the result names to `names(x)` (or `rownames(x)` for a data frame argument). When names is a character vector, we set the result names to this vector of names.

Similarly, when filter is `NULL`, we set the result text filter to `text_filter(x)`. When filter is non-NULL missing, we set the result text filter to this value. In either case, if there are additional
names arguments, then we override the filter properties specified by the names of these arguments with the new values given.

Note that the special handling for the names of the object is different from the other R conversion functions (as.numeric, as.character, etc.), which drop the names.

as_corpus_text is generic: you can write methods to handle specific classes of objects.

Value

as_corpus_text attempts to coerce its argument to text type and set its names and text_filter properties; it strips all other attributes.

is_corpus_text returns TRUE or FALSE depending on whether its argument is of text type or not.

See Also

as_utf8, text_filter, read_ndjson.

Examples

as_corpus_text("hello, world!")

as_corpus_text(c(a = "goodnight", b = "moon")) # keeps names

# set a filter property
as_corpus_text(c(a = "goodnight", b = "moon"), stemmer = "english")

is_corpus_text("hello") # FALSE, "hello" is character, not text

---

federalist

The Federalist Papers

Description

The Federalist Papers comprise 85 articles published under the pseudonym “Publius” in New York newspapers between 1787 and 1788, written to convince residents to ratify the Constitution. John Jay wrote 5 papers, while Alexander Hamilton and James Madison wrote the remaining 80. Between the last two authors there are conflicting accounts of which author wrote which paper. Most sources agree on the authorships of 65 papers (51 by Hamilton and 14 by Madison), but 15 papers are in dispute.

In one of the earliest examples of statistical text analysis, F. Mosteller and D. L. Wallace used a form of Naive Bayes classification to identify the authorships of the 15 disputed papers, finding strong evidence that Madison was the author of all of the disputed papers.

Usage

federalist

Format

A data frame with 85 rows, one for each paper.
Source

http://www.gutenberg.org/ebooks/18

References


---

Project Gutenberg Corpora

Description

Get a corpus of texts from Project Gutenberg.

Usage

gutenberg_corpus(ids, filter = NULL, mirror = NULL, verbose = TRUE, ...)

Arguments

ids  
an integer vector of requested Gutenberg text IDs.

filter  
a text filter to set on the corpus.

mirror  
a character string URL for the Gutenberg mirror to use, or NULL to determine automatically.

verbose  
a logical scalar indicating whether to print progress updates to the console.

...  
additional arguments passed to `as_corpus`.

Details

gutenberg_corpus downloads a set of texts from Project Gutenberg, creating a corpus with the texts as rows. You specify the texts for inclusion using their Project Gutenberg IDs, passed to the function in the `ids` argument.

You can search for Project Gutenberg texts and get their IDs using the `gutenberg_works` function from the gutenbergr package.

Value

A corpus (data frame) with three columns: "title", "author", and "text".

See Also

corpus_frame.

Examples

# get the texts of George Eliot's novels
## Not run: eliot <- gutenberg_corpus(c(145, 550, 6688))
new_stemmer

**Description**

Make a stemmer from a set of (term, stem) pairs.

**Usage**

```r
new_stemmer(term, stem, default = NULL, duplicates = "first",
vectorize = TRUE)
```

**Arguments**

- `term` character vector of terms to stem.
- `stem` character vector the same length as `term` with entries giving the corresponding stems.
- `default` if non-NULL, a default value to use for terms that do not have a stem; `NULL` specifies that such terms should be left unchanged.
- `duplicates` action to take for duplicates in the `term` list. See ‘Details’.
- `vectorize` whether to produce a vectorized stemmer that accepts and returns vector arguments.

**Details**

Giving a list of terms and a corresponding list of stems, this produces a function that maps terms to their corresponding entry. If `default = NULL`, then values absent from the `term` argument get left as-is; otherwise, they get replaced by the `default` value.

The `duplicates` argument indicates the action to take if there are duplicate entries in the `term` argument:

- `duplicates = "first"` take the first matching entry in the `stem` list.
- `duplicates = "last"` take the last matching entry in the `stem` list.
- `duplicates = "omit"` use the default value for duplicated terms.
- `duplicates = "fail"` raise an error if there are duplicated terms.

**Value**

By default, with `vectorize = TRUE`, the resulting stemmer accepts a character vector as input and returns a character vector of the same length with entries giving the stems of the corresponding input entries.

Setting `vectorize = FALSE` gives a function that accepts a single input and returns a single output. This can be more efficient when used as part of a `text_filter`. 
See Also

stem_snowball, text_filter.text_tokens.

Examples

# map uppercase to lowercase, leave others unchanged
stemmer <- new_stemmer(LETTERS, letters)
stemmer(c("A", "E", "I", "O", "U", "1", "2", "3"))

# map uppercase to lowercase, drop others
stemmer <- new_stemmer(LETTERS, letters, default = NA)
stemmer(c("A", "E", "I", "O", "U", "1", "2", "3"))

print.corpus_frame

Corpus Data Frame Printing

Description

Printing and formatting corpus data frames.

Usage

## S3 method for class 'corpus_frame'
print(x, rows = 20L, chars = NULL, digits = NULL,
quote = FALSE, na.print = NULL, print.gap = NULL,right = FALSE,
row.names = TRUE, max = NULL, display = TRUE, ...)

## S3 method for class 'corpus_frame'
format(x, chars = NULL, na.encode = TRUE, quote = FALSE,
na.print = NULL, print.gap = NULL, ..., justify = "none")

Arguments

x              data frame object to print or format.
rows           integer scalar giving the maximum number of rows to print before truncating the output. A negative or missing value indicates no upper limit.
chars          maximum number of character units to display; see utf8_format.
digits         minimal number of significant digits; see print.default.
quote          logical scalar indicating whether to put surrounding double-quotes ('"') around character strings and escape internal double-quotes.
a.print        character string (or NULL) indicating the encoding for NA values. Ignored when na.encode is FALSE.
print.gap      non-negative integer (or NULL) giving the number of spaces in gaps between columns; set to NULL or 1 for a single space.
right          logical indicating whether to right-align columns (ignored for text, character, and factor columns).
row.names logical indicating whether to print row names, or a character vector giving alternate row names to display.
max maximum number of entries to print; defaults to getOption("max.print").
display logical scalar indicating whether to optimize the printing for display, not byte-for-byte data transmission; see utf8_encode.
justify justification; one of "left", "right", "centre", or "none". Can be abbreviated.
na.encode logical scalar indicating whether to encode NA values as character strings.
... further arguments passed to or from other methods.

Details

The "corpus_frame" class is a subclass of "data.frame", overriding the default print and format methods. To apply this class to a data frame, set class to c("corpus_frame", "data.frame").

Corpus frame printing left-justifies character and text columns, truncates the output, and displays emoji on Mac OS.

See Also
corpus_frame, print.data.frame, utf8_print

Examples

# default data frame printing
x <- data.frame(text = c("hello world", intToUtf8(0x1f638 + 0:3), letters))
print(x)

# corpus frame printing
y <- x
class(y) <- c("corpus_frame", "data.frame")
print(y)

print(y, 10) # change truncation limit

Description

Read data from a file in newline-delimited JavaScript Object Notation (NDJSON) format.

Usage

read_ndjson(file, mmap = FALSE, simplify = TRUE, text = NULL)
Arguments

file the name of the file which the data are to be read from, or a connection (unless mmap is TRUE, see below). The data should be encoded as UTF-8, and each line should be a valid JSON value.

mmap whether to memory-map the file instead of reading all of its data into memory simultaneously. See the ‘Memory mapping’ section.

simplify whether to attempt to simplify the type of the return value. For example, if each line of the file stores an integer, if simplify is set to TRUE then the return value will be an integer vector rather than a corpus_json object.

text a character vector of string fields to interpret as text instead of character, or NULL to interpret all strings as character.

Details

This function is the recommended means of reading data for processing by the corpus package.

When the text argument is non-NULL string data fields with names indicated by this argument are decoded as text values, not as character values.

Value

In the default usage, with argument simplify = TRUE, when the lines of the file are records (JSON object literals), the return value from read_ndjson is a data frame with class c("corpus_frame", "data.frame"). With simplify = FALSE, the result is a corpus_json object.

Memory mapping

When you specify mmap = TRUE, the function memory-maps the file instead of reading it into memory directly. In this case, the file argument must be a character string giving the path to the file, not a connection object. When you memory-map the file, the operating system reads data into memory only when it is needed, enabling you to transparently process large data sets that do not fit into memory.

In terms of memory usage, enabling mmap = TRUE reduces the footprint for corpus_json and corpus_text objects; native R objects (character, integer, list, logical, and numeric) get fully deserialized to memory and produce identical results regardless of whether mmap is TRUE or FALSE. To process a large text corpus with a text field named "text", you should set text = "text" and mmap = TRUE. Or, to reduce the memory footprint even further, set simplify = FALSE and mmap = TRUE.

One danger in memory-mapping is that if you delete the file after calling read_ndjson but before processing the data, then the results will be undefined, and your computer may crash. (On POSIX-compliant systems like Mac OS and Linux, there should be no ill effects to deleting the file. On recent versions of Windows, the system will not allow you to delete the file as long as the data is active.)

Another danger in memory-mapping is that if you serialize a corpus_json object or derived corpus_text object using saveRDS or another similar function, and then you deserialize the object, R will attempt to create a new memory-map using the file argument passed to the original read_ndjson call. If file is a relative path, then your working directory at the time of deserialization must agree with
your working directory at the time of the `read_ndjson` call. You can avoid this situation by specifying an absolute path as the `file` argument (the `normalizePath` function will convert a relative to an absolute path).

**See Also**

`as_corpus_text`, `as_utf8`.

**Examples**

```r
data$a
data$b
data$nested.c
data$nested.d
rm("data")
invisible(gc()) # force the garbage collector to release the memory-map
file.remove(file)
```

---

### sentiment_afinn

**AFINN Sentiment Lexicon**

#### Description

The AFINN lexicon is a list of English terms manually rated for valence with an integer between -5 (negative) and +5 (positive) by Finn Årup Nielsen between 2009 and 2011.

The original lexicon contains some multi-word phrases, but they are excluded here.

The original lexicon is distributed under the Open Database License (ODbL) v1.0. You are free to share, create works from, and adapt the lexicon, as long as you attribute the original lexicon in your work. If you adapt the lexicon, you must keep the adapted lexicon open and apply a similar license.

#### Usage

`sentiment_afinn`

#### Format

A data frame with one row for each term
stem_snowball

Description

Stem a set of terms using one of the algorithms provided by the Snowball stemming library.

Usage

stem_snowball(x, algorithm = "en")

Arguments

x character vector of terms to stem.
algorithm stemming algorithm; see ‘Details’ for the valid choices.

Details

Apply a Snowball stemming algorithm to a vector of input terms, x, returning the result in a character vector of the same length with the same names.

The algorithm argument specifies the stemming algorithm. Valid choices include the following: "ar" ("arabic"), "da" ("danish"), "de" ("german"), "en" ("english"), "es" ("spanish"), "fi" ("finnish"), "fr" ("french"), "hu" ("hungarian"), "it" ("italian"), "nl" ("dutch"), "no" ("norwegian"), "pt" ("portuguese"), "ro" ("romanian"), "ru" ("russian"), "sv" ("swedish"), "ta" ("tamil"), "tr" ("turkish"), and "porter". Setting algorithm = NULL gives a stemmer that returns its input unchanged.

The function only stems single-word terms of kind "letter"; it leaves other inputs (multi-word terms, and terms of kind "number", "punct", and "symbol") unchanged.

The Snowball stemming library provides the underlying implementation. The wordstem function from the SnowballC package provides a similar interface, but that function applies the algorithm to all input terms, regardless of the kind of the term.

Value

A character vector the same length and names as the input, x, with entries containing the corresponding stems.
**stopwords**

**See Also**

`new_stemmer`, `text_filter`.

**Examples**

```r
# apply english stemming algorithm; don't stem non-letter terms
stem_snowball(c("win", "winning", "winner", "#winning"))

# compare with SnowballC, which stems all kinds, not just letter
## Not run: SnowballC::wordStem(c("win", "winning", "winner", "#winning"), "en")
```

---

**stopwords**

<table>
<thead>
<tr>
<th>Stop Words</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

**Description**

Lists of common function words (‘stop’ words).

**Usage**

- `stopwords_da`
- `stopwords_de`
- `stopwords_en`
- `stopwords_es`
- `stopwords_fi`
- `stopwords_fr`
- `stopwords_hu`
- `stopwords_it`
- `stopwords_nl`
- `stopwords_no`
- `stopwords_pt`
- `stopwords_ru`
- `stopwords_sv`

**Format**

A character vector of unique stop words.

**Details**

The `stopwords_` objects are character vectors of case-folded ‘stop’ words. These are common function words that often get discarded before performing other text analysis tasks.

There are lists available for the following languages: Danish (`stopwords_da`), Dutch (`stopwords_nl`), English (`stopwords_en`), Finnish (`stopwords_fi`), French (`stopwords_fr`), German (`stopwords_de`), Hungarian (`stopwords_hu`), Italian (`stopwords_it`), Norwegian (`stopwords_no`), Portuguese (`stopwords_pt`), Russian (`stopwords_ru`), Spanish (`stopwords_es`), and Swedish (`stopwords_sv`).
These built-in word lists are reasonable defaults, but they may require further tailoring to suit your particular task. The original lists were compiled by the Snowball stemming project. Following the Quanteda text analysis software, we have tailored the original lists by adding the word "will" to the English list.

See Also
text_filter

term_matrix

Description
Tokenize a set of texts and compute a term frequency matrix.

Usage

```r
term_matrix(x, filter = NULL, ngrams = NULL, select = NULL,
             group = NULL, transpose = FALSE, ...)

term_counts(x, filter = NULL, ngrams = NULL, select = NULL,
             group = NULL, ...)
```

Arguments

- `x` a text vector to tokenize.
- `filter` if non-NULL, a text filter to use instead of the default text filter for `x`.
- `ngrams` an integer vector of n-gram lengths to include, or NULL to use the select argument to determine the n-gram lengths.
- `select` a character vector of terms to count, or NULL to count all terms that appear in `x`.
- `group` if non-NULL, a factor, character string, or integer vector the same length of `x` specifying the grouping behavior.
- `transpose` a logical value indicating whether to transpose the result, putting terms as rows instead of columns.
- `...` additional properties to set on the text filter.

Details

term_matrix tokenizes a set of texts and computes the occurrence counts for each term, returning the result as a sparse matrix (texts-by-terms). term_counts returns the same information, but in a data frame.

If `ngrams` is non-NULL, then multi-type n-grams are included in the output for all lengths appearing in the ngrams argument. If `ngrams` is NULL but `select` is non-NULL, then all n-grams appearing in the select set are included. If both `ngrams` and `select` are NULL, then only unigrams (single type terms) are included.
If group is NULL, then the output has one set of term counts for each input text. Otherwise, we convert group to a factor and compute one set of term counts for each level. Texts with NA values for group get skipped.

Value

term_matrix with transpose = FALSE returns a sparse matrix in "dgCMatrix" format with one column for each term and one row for each input text or (if group is non-NULL) for each grouping level. If filter$select is non-NULL, then the column names will be equal to filter$select. Otherwise, the columns are assigned in arbitrary order.

term_matrix with transpose = TRUE returns the transpose of the term matrix, in "dgCMatrix" format.

term_counts with group = NULL returns a data frame with one row for each entry of the term matrix, and columns "text", "term", and "count" giving the text ID, term, and count. The "term" column is a factor with levels equal to the selected terms. The "text" column is a factor with levels equal to names(as_corpus_text(x)); calling as.integer on the "text" column converts from the factor values to the integer row index in the term matrix.

term_counts with group non-NULL behaves similarly, but the result instead has columns named "group", "term", and "count", with "group" giving the grouping level, as a factor.

See Also

text_tokens, term_stats.

Examples

text <- c("A rose is a rose is a rose.",
          "A Rose is red, a violet is blue!",
          "A rose by any other name would smell as sweet.")
term_matrix(text)

# select certain terms
term_matrix(text, select = c("rose", "red", "violet", "sweet"))

# specify a grouping factor
term_matrix(text, group = c("Good", "Bad", "Good"))

# include higher-order n-grams
term_matrix(text, ngrams = 1:3)

# select certain multi-type terms
term_matrix(text, select = c("a rose", "a violet", "sweet", "smell"))

# transpose the result
term_matrix(text, ngrams = 1:2, transpose = TRUE)[1:10, ] # first 10 rows

# data frame
head(term_counts(text), n = 10) # first 10 rows

# with grouping
term_counts(text, group = c("Good", "Bad", "Good"))

# taking names from the input
term_counts(c(a = "One sentence.", b = "Another", c = "!!"))

<table>
<thead>
<tr>
<th>term_stats</th>
<th>Term Statistics</th>
</tr>
</thead>
</table>

**Description**

Tokenize a set of texts and tabulate the term occurrence statistics.

**Usage**

```r
term_stats(x, filter = NULL, ngrams = NULL,  
            min_count = NULL, max_count = NULL,  
            min_support = NULL, max_support = NULL, types = FALSE,  
            subset, ...)
```

**Arguments**

- `x` a text vector to tokenize.
- `filter` if non-NULL, a text filter to to use instead of the default text filter for `x`.
- `ngrams` an integer vector of n-gram lengths to include, or NULL for length-1 n-grams only.
- `min_count` a numeric scalar giving the minimum term count to include in the output, or NULL for no minimum count.
- `max_count` a numeric scalar giving the maximum term count to include in the output, or NULL for no maximum count.
- `min_support` a numeric scalar giving the minimum term support to include in the output, or NULL for no minimum support.
- `max_support` a numeric scalar giving the maximum term support to include in the output, or NULL for no maximum support.
- `types` a logical value indicating whether to include columns for the types that make up the terms.
- `subset` logical expression indicating elements or rows to keep: missing values are taken as false.
- `...` additional properties to set on the text filter.

**Details**

term_stats tokenizes a set of texts and computes the occurrence counts and supports for each term. The ‘count’ is the number of occurrences of the term across all texts; the ‘support’ is the number of texts containing the term. Each appearance of a term increments its count by one. Likewise, an appearance of a term in text i increments its support once, not for each occurrence in the text.

To include multi-type terms, specify the designed term lengths using the `ngrams` argument.
**Value**

A data frame with columns named term, count, and support, with one row for each appearing term. Rows are sorted in descending order according to support and then count, with ties broken lexicographically by term, using the character ordering determined by the current locale (see *Comparison* for details).

If types = TRUE, then the result also includes columns named type1, type2, etc. for the types that make up the term.

**See Also**

*text_tokens, term_matrix.*

**Examples**

```r
term_stats("A rose is a rose is a rose."")

# remove punctuation and English stop words
term_stats("A rose is a rose is a rose.",
  text_filter(drop_symbol = TRUE, drop = stopwords_en))

# unigrams, bigrams, and trigrams
term_stats("A rose is a rose is a rose.", ngrams = 1:3)

# also include the type information
term_stats("A rose is a rose is a rose.", ngrams = 1:3, types = TRUE)
```

---

**Description**

Get or specify the process by which text gets transformed into a sequence of tokens or sentences.

**Usage**

```r
text_filter(x = NULL, ...)
text_filter(x) <- value

## S3 method for class 'corpus_text'
text_filter(x = NULL, ...)

## S3 method for class 'data.frame'
text_filter(x = NULL, ...)

## Default S3 method:
text_filter(x = NULL, ...,
  map_case = TRUE, map_quote = TRUE,
```

**Text Filters**
remove_ignorable = TRUE,
combine = NULL,
stemmer = NULL, stem_dropped = FALSE,
stem_except = NULL,
drop_letter = FALSE, drop_number = FALSE,
drop_punct = FALSE, drop_symbol = FALSE,
drop = NULL, drop_except = NULL,
connector = ".",
sent_crlf = FALSE,
sent_suppress = corpus::abbreviations_en)

Arguments

x text or corpus object.
value text filter object, or NULL for the default.
... further arguments passed to or from other methods.
map_case a logical value indicating whether to apply Unicode case mapping to the text. For most languages, this transformation changes uppercase characters to their lowercase equivalents.
map_quote a logical value indicating whether to replace curly single quotes and other Unicode apostrophe characters with ASCII apostrophe (U+0027).
remove_ignorable a logical value indicating whether to remove Unicode "default ignorable" characters like zero-width spaces and soft hyphens.
combine a character vector of multi-word phrases to combine, or NULL; see 'Combining words'.
stemmer a character value giving the name of a Snowball stemming algorithm (see stem_snowball for choices), a custom stemming function, or NULL to leave words unchanged.
stem_dropped a logical value indicating whether to stem words in the "drop" list.
stem_except a character vector of exception words to exempt from stemming, or NULL. If left unspecified, stem_except is set equal to the drop argument.
drop_letter a logical value indicating whether to replace "letter" tokens (cased letters, kana, ideographic, letter-like numeric characters and other letters) with NA.
drop_number a logical value indicating whether to replace "number" tokens (decimal digits, words appearing to be numbers, and other numeric characters) with NA.
drop_punct a logical value indicating whether to replace "punct" tokens (punctuation) with NA.
drop_symbol a logical value indicating whether to replace "symbol" tokens (emoji, math, currency, URLs, and other symbols) with NA.
drop a character vector of types to replace with NA, or NULL.
drop_except a character of types to exempt from the drop rules specified by the drop_letter, drop_number, drop_punct, drop_symbol, and drop arguments, or NULL.
connector a character to use as a connector in lieu of white space for types that stem to multi-word phrases.
text_filter

sent_crlf a logical value indicating whether to break sentences on carriage returns or line feeds.

sent_suppress a character vector of sentence break suppressions.

Details
The set of properties in a text filter determine the tokenization and sentence breaking rules. See the
documentation for text_tokens and text_split for details on the tokenization process.

Value
text_filter retrieves an objects text filter, optionally with modifications to some of its properties.
text_filter<- sets an object's text filter. Setting the text filter on a character object is not al-
lowed; the object must have type "corpus_text" or be a data frame with a "text" column of type
"corpus_text".

See Also
as_corpus_text, text_tokens, text_split, abbreviations, stopwords.

Examples

# text filter with default options set
text_filter()

# specify some options but leave others unchanged
f <- text_filter(map_case = FALSE, drop = stopwords_en)

# set the text filter property
x <- as_corpus_text(c("Marnie the Dog is #1 on the internet."))
text_filter(x) <- f
text_tokens(x) # by default, uses x's text_filter to tokenize

# change a filter property
f2 <- text_filter(x, map_case = TRUE)
# equivalent to:
# f2 <- text_filter(x)
# f2$map_case <- TRUE
text_tokens(x, f2) # override text_filter(x)

# setting text_filter on a data frame is allowed if it has a
# column names "text" of type "corpus_text"
d <- data.frame(text = x)
text_filter(d) <- f2
text_tokens(d)

# but you can't set text filters on character objects
y <- "hello world"
## Not run: text_filter(y) <- f2 # gives an error
text_locate

Description
Look for instances of one or more terms in a set of texts.

Usage
text_locate(x, terms, filter = NULL, ...)
text_count(x, terms, filter = NULL, ...)
text_detect(x, terms, filter = NULL, ...)
text_match(x, terms, filter = NULL, ...)
text_sample(x, terms, size = NULL, filter = NULL, ...)
text_subset(x, terms, filter = NULL, ...)

Arguments
x a text or character vector.
terms a character vector of search terms.
filter if non-NULL, a text filter to to use instead of the default text filter for x.
size the maximum number of results to return, or NULL.
... additional properties to set on the text filter.

Details
text_locate finds all instances of the search terms in the input text, along with their contexts.
text_count counts the number of search term instances in each element of the text vector.
text_detect indicates whether each text contains at least one of the search terms.
text_match reports the matching instances as a factor variable with levels equal to the terms argument.
text_subset returns the texts that contain the search terms.
text_sample returns a random sample of the results from text_locate, in random order. This is useful for hand-inspecting a subset of the text_locate matches.
Value

text_count and text_detect return a numeric vector and a logical vector, respectively, with length equal to the number of input texts and names equal to the text names.

text_locate and text_sample both return a data frame with one row for each search result and columns named 'text', 'before', 'instance', and 'after'. The 'text' column gives the name of the text containing the instance; 'before' and 'after' are text vectors giving the text before and after the instance. The 'instance' column gives the token or tokens matching the search term.

text_match returns a data frame for one row for each search result, with columns names 'text' and 'term'. Both columns are factors. The 'text' column has levels equal to the text labels, and the 'term' column has levels equal to terms argument.

text_subset returns the subset of texts that contain the given search terms. The resulting has its text_filter set to the passed-in filter argument.

See Also

term_stats, term_matrix.

Examples

text <- c("Rose is a rose is a rose is a rose. ",
          "A rose by any other name would smell as sweet.",
          "Snow White and Rose Red")

text_count(text, "rose")
text_detect(text, "rose")
text_locate(text, "rose")
text_match(text, "rose")
text_sample(text, "rose", 3)
text_subset(text, "a rose")

# search for multiple terms
text_locate(text, c("rose", "rose red", "snow white"))

---

**text_split**

**Segmenting Text**

**Description**

Segment text into smaller units.

**Usage**

text_split(x, units = "sentences", size = 1, filter = NULL, ...)
text_nsentence(x, filter = NULL, ...)

Arguments

- **x**: a text or character vector.
- **units**: the block size units, either "sentences" or "tokens".
- **size**: the block size, a positive integer giving the maximum number of units per block.
- **filter**: if non-NULL, a text filter to use instead of the default text filter for `x`.
- **...**: additional properties to set on the text filter.

Details

text_split splits text into roughly evenly-sized blocks, measured in the specified units. When `units` = "sentences", units are sentences; when `units` = "tokens", units are non-NA tokens. The size parameter specifies the maximum block size.

When the minimum block size does not evenly divide the number of total units in a text, the block sizes will not be exactly equal. However, it will still be the case that no block will have more than one unit more than any other block. The extra units get allocated to the first segments in the split.

Sentences and tokens are defined by the `filter` argument. The documentation for text_tokens describes the tokenization rules. For sentence boundaries, see the ‘Sentences’ section below.

Value

text_split returns a data frame with three columns named `parent`, `index`, and `text`, and one row for each text block. The columns are as follows:

1. The `parent` column is a factor. The levels of this factor are the names of `as_corpus_text(x)`.
   Calling `as.integer` on the parent column gives the indices of the parent texts for the parent text for each sentence.
2. The `index` column gives the integer index of the sentence in its parent.
3. The `text` value is the text of the block, a value of type `corpus_text` (not a character vector).

text_nsentence returns a numeric vector with the same length as `x` with each element giving the number of sentences in the corresponding text.

Sentences

Sentences are defined according to a tailored version of the boundaries specified by Unicode Standard Annex #29, Section 5.

The UAX 29 sentence boundaries handle Unicode correctly and they give reasonable behavior across a variety of languages, but they do not handle abbreviations correctly and by default they treat carriage returns and line feeds as paragraph separators, often leading to incorrect breaks. To get around these shortcomings, the text filter allows tailoring the UAX 29 rules using the `sent_crlf` and the `sent_suppress` properties.

The UAX 29 rules break after full stops (periods) whenever they are followed by uppercase letters. Under these rules, the text “I saw Mr. Jones today.” gets split into two sentences. To get around this, we allow a `sent_suppress` property, a list of sentence break suppressions which, when followed by uppercase characters, do not signal the end of a sentence.
The UAX 29 rules also specify that a carriage return (CR) or line feed (LF) indicates the end of of a sentence, so that "A split
sentence." gets split into two sentences. This often leads to incorrect breaks, so by default, with sent_crlf = FALSE, we deviate from the UAX 29 rules and we treat CR and LF like spaces. To break sentences on CRLF, CR, and LF, specify sent_crlf = TRUE.

See Also
text_tokens, text_filter.

Examples

text <- c("I saw Mr. Jones today.",
           "Split across\na line.",
           "She asked 'do you really mean that?' and I said 'yes.'")

# split text into sentences
text_split(text, units = "sentences")

# get the number of sentences
text_nsentence(text)

# disable the default sentence suppressions
text_split("I saw Mr. Jones today.", units = "sentences", filter = NULL)

# break on CR and LF
text_split("Split across\na line.", units = "sentences",
           filter = text_filter(sent_crlf = TRUE))

# 2-sentence blocks
text_split(c("What. Are. You. Doing???",
           "She asked 'do you really mean that?' and I said 'yes.'"),
           units = "sentences", size = 2)

# 4-token blocks
text_split(c("What. Are. You. Doing???",
           "She asked 'do you really mean that?' and I said 'yes.'"),
           units = "sentences", size = 4)

# blocks are approximately evenly sized; 'size' gives maximum size
text_split(paste(letters, collapse = " "), "tokens", 4)
text_split(paste(letters, collapse = " "), "tokens", 16)
Usage

\text{text_stats}(x, \text{filter} = \text{NULL, } \ldots)

Arguments

\begin{itemize}
  \item \texttt{x} \hspace{1cm} \text{a text corpus.}
  \item \texttt{filter} \hspace{1cm} \text{if non-NULL, a text filter to to use instead of the default text filter for \texttt{x}.}
  \item \texttt{\ldots} \hspace{1cm} \text{additional properties to set on the text filter.}
\end{itemize}

Details

\text{text_stats} reports descriptive statistics for a set of texts: the number of tokens, unique types, and sentences.

Value

A data frame with columns named \texttt{tokens}, \texttt{types}, and \texttt{sentences}, with one row for each text.

See Also

\texttt{text_filter, term_stats}.

Examples

\begin{verbatim}
\text{text_stats(c("A rose is a rose is a rose.",
              "A Rose is red. A violet is blue!
              "'))}
\end{verbatim}

\begin{verbatim}
\begin{verbatim}
\begin{verbatim}
\begin{verbatim}
\begin{verbatim}
Details
text_sub extracts token subsequences from a set of texts. The start and end arguments specifying the positions of the subsequences within the parent texts, as an inclusive range. Negative indices are interpreted as counting from the end of the text, with -1L referring to the last element.

Value
A text vector with the same length and names as x, with the desired subsequences.

See Also
text_tokens, text_ntoken.

Examples
x <- as_corpus_text(c("A man, a plan.", "A "canal"?", "Panama!"),
drop_punct = TRUE)

# entire text
text_sub(x, 1, -1)

# first three elements
text_sub(x, 1, 3)

# last two elements
text_sub(x, -2, -1)
Details

text_tokens splits texts into token sequences. Each token is an instance of a particular type. This operation proceeds in a series of stages, controlled by the filter argument:

1. First, we segment the text into words and spaces using the boundaries defined by Unicode Standard Annex #29, Section 4, with special handling for @mentions, #hashtags, and URLs.
2. Next, we normalize the words by applying the character mappings indicated by the map_case, map_quote, and remove_ignorable properties. We replace sequences of spaces by a space (U+0020). At the end of the second stage, we have segmented the text into a sequence of normalized words and spaces, in Unicode composed normal form (NFC).
3. In the third stage, if the combine property is non-NULL, we scan the word sequence from left to right, searching for the longest possible match in the combine list. If a match exists, we replace the word sequence with a single token for that term; otherwise, we leave the word as-is. We drop spaces at this point, unless they are part of a multi-word term. See the ‘Combining words’ section below for more details.
4. Next, if the stemmer property is non-NULL, we apply the indicated stemming algorithm to each word that does not match one of the elements of the stem_except character vector. Terms that stem to NA get dropped from the sequence.
5. After stemming, we categorize each remaining token as "letter", "number", "punct", or "symbol" according to the first character in the word. For words that start with extenders like underscore (_), use the first non-extender to classify it.
6. If any of drop_letter, drop_number, drop_punct, or drop_symbol are TRUE, then we drop the tokens in the corresponding categories. We also drop any terms that match an element of the drop character vector. We can add exceptions to the drop rules by specifying a non-NULL value for the drop_except property: drop_except is a character vector, then we we restore tokens that match elements of vector to their values prior to dropping.
7. Finally, we replace sequences of white-space in the terms with the specified connector, which defaults to a low line character (_, U+005F).

Multi-word terms specified by the combine property can be specified as tokens, prior to normalization. Terms specified by the stem_except, drop, and drop_except need to be normalized and stemmed (if stemmer is non-NULL). Thus, for example, if map_case = TRUE, then a token filter with combine = "mx." produces the same results as a token filter with combine = "mx.". However, drop = "mx." behaves different from drop = "mx.".

Value

text_tokens returns a list of the same length as x, with the same names. Each list item is a character vector with the tokens for the corresponding element of x.
text_ntoken returns a numeric vector the same length as x, with each element giving the number of tokens in the corresponding text.

Combining words

The combine property of a text_filter enables transformations that combine two or more words into a single token. For example, specifying combine = "new york" will cause consecutive instances of the words new and york to get replaced by a single token, new york.
See Also

stopwords, text_filter, text_types.

Examples

text_tokens("The quick ('brown') fox can't jump 32.3 feet, right?")

# count tokens:
text_ntoken("The quick ('brown') fox can't jump 32.3 feet, right?")

# don't change case or quotes:
f <- text_filter(map_case = FALSE, map_quote = FALSE)
text_tokens("The quick ('brown') fox can't jump 32.3 feet, right?", f)

# drop common function words ('stop' words):
text_tokens("Able was I ere I saw Elba.",
    text_filter(drop = stopwords_en))

# drop numbers, with some exceptions:
text_tokens("0, 1, 2, 3, 4, 5",
    text_filter(drop_number = TRUE,
        drop_except = c("0", "2", "4")))

# apply stemming...
text_tokens("Mary is running", text_filter(stemmer = "english"))

# ...except for certain words
text_tokens("Mary is running",
    text_filter(stemmer = "english", stem_except = "mary"))

# default tokenization
text_tokens("Ms. Jones")

# combine abbreviations
text_tokens("Ms. Jones", text_filter(combine = abbreviations_en))

# add custom combinations
text_tokens("Ms. Jones is from New York City, New York.",
    text_filter(combine = c(abbreviations_en,
        "new york", "new york city")))

---

<table>
<thead>
<tr>
<th>text_types</th>
<th>Text Type Sets</th>
</tr>
</thead>
</table>

Description

Get or measure the set of types (unique token values).
### Usage

```r
text_types(x, filter = NULL, collapse = FALSE, ...)
text_notype(x, filter = NULL, collapse = FALSE, ...)
```

### Arguments

- **x**: a text or character vector.
- **filter**: if non-NULL, a text filter to use instead of the default text filter for `x`.
- **collapse**: a logical value indicating whether to collapse the aggregation over all rows of the input.
- **...**: additional properties to set on the text filter.

### Details

`text_notype` counts the number of unique types in each text; `text_types` returns the set of unique types, as a character vector. Types are determined according to the `filter` argument.

### Value

If `collapse = FALSE`, then `text_notype` produces a numeric vector with the same length and names as the input text, with the elements giving the number of units in the corresponding texts. For `text_types`, the result is a list of character vector with each vector giving the unique types in the corresponding text, ordered according to the `sort` function.

If `collapse = TRUE`, then we aggregate over all rows of the input. In this case, `text_notype` produces a scalar indicating the number of unique types in `x`, and `text_types` produces a character vector with the unique types.

### See Also

`text_filter`, `text_tokens`.

### Examples

```r
text <- c("I saw Mr. Jones today.",
           "Split across\na line.",
           "She asked 'do you really mean that?' and I said 'yes.'")

# count the number of unique types
text_notype(text)
text_notype(text, collapse = TRUE)

# get the type sets
text_types(text)
text_types(text, collapse = TRUE)
```
Index

*Topic classes
  corpus_frame, 4
corpus_text, 6
*Topic datasets
  abbreviations, 3
  affect_wordnet, 3
  federalist, 7
  sentiment_afinn, 13
  stopwords, 15
*Topic file
  read_ndjson, 11
*Topic package
corpus_package, 2
  abbreviations, 3, 21
  abbreviations_de (abbreviations), 3
  abbreviations_en (abbreviations), 3
  abbreviations_es (abbreviations), 3
  abbreviations_fr (abbreviations), 3
  abbreviations_it (abbreviations), 3
  abbreviations_pt (abbreviations), 3
  abbreviations_ru (abbreviations), 3
  affect_wordnet, 3
  as_corpus_frame (corpus_frame), 4
  as_corpus_text, 2, 5, 13, 21
  as_corpus_text (corpus_text), 6
  as_utf8, 7, 13

Comparison, 19
corpus (corpus-package), 2
corpus-package, 2
corpus_frame, 4, 8, 11
corpus_json (read_ndjson), 11
corpus_text, 5, 6, 24
corpus_text_filter (text_filter), 19
data.frame, 5
federalist, 7
format.corpus_frame
  (print.corpus_frame), 10
gutenberg_corpus, 8
is_corpus_frame (corpus_frame), 4
is_corpus_text (corpus_text), 6
new_stemmer, 9, 15
normalizePath, 13
print.corpus_frame, 5, 10
print.data.frame, 11
print.default, 10
read_ndjson, 2, 5–7, 11
saveDS, 12
sentences (text_split), 23
sentiment_afinn, 13
sort, 30
stem_snowball, 10, 14, 20
stopwords, 15, 21, 29
stopwords_da (stopwords), 15
stopwords_de (stopwords), 15
stopwords_en (stopwords), 15
stopwords_es (stopwords), 15
stopwords_fr (stopwords), 15
stopwords_hu (stopwords), 15
stopwords_it (stopwords), 15
stopwords_nl (stopwords), 15
stopwords_no (stopwords), 15
stopwords_pt (stopwords), 15
stopwords_ru (stopwords), 15
stopwords_sv (stopwords), 15
term_counts (term_matrix), 16
term_matrix, 2, 16, 19, 23
term_stats, 2, 17, 18, 23, 26
text_count, 2
text_count (text_locate), 22
text_detect, 2
text_detect (text_locate), 22
text_filter, 2, 3, 6, 7, 9, 10, 15, 16, 19, 25, 26, 29, 30

   text_filter <= (text_filter), 19

  text_locate, 2, 22

   text_match (text_locate), 22

  text_nsent (text_split), 23

  text_ntoken, 27

   text_ntoken (text_tokens), 27

  text_notype (text_types), 29

  text_sample (text_locate), 22

  text_split, 2, 21, 23

  text_stats, 25

  text_sub, 26

  text_subset (text_locate), 22

  text_tokens, 2, 10, 17, 19, 21, 24, 25, 27, 27, 30

  text_types, 29, 29

utf8_format, 10

utf8_print, 11