Package ‘quanteda’

November 26, 2019

Version 1.5.2
Title Quantitative Analysis of Textual Data
Description A fast, flexible, and comprehensive framework for quantitative text analysis in R. Provides functionality for corpus management, creating and manipulating tokens and ngrams, exploring keywords in context, forming and manipulating sparse matrices of documents by features and feature co-occurrences, analyzing keywords, computing feature similarities and distances, applying content dictionaries, applying supervised and unsupervised machine learning, visually representing text and text analyses, and more.
License GPL-3
Depends R (>= 3.1.0), methods
Imports data.table (>= 1.9.6), extrafont, fastmatch, ggplot2 (>= 2.2.0), ggrepel, lubridate, magrittr, Matrix (>= 1.2), network, RSpectra, Rcpp (>= 0.12.12), RcppParallel, sna, SnowballC, spacyr, stopwords, stringi, xml2, yaml, proxyC (>= 0.1.4)
LinkingTo Rcpp, RcppParallel, RcppArmadillo (>= 0.7.600.1.0)
Suggests ca, dplyr, DT, e1071, entropy, ExPosition, lda, Isa, proxy, purrr, RColorBrewer, rmarkdown, slam, spelling, stm, svs, testthat, text2vec, tibble, tidytext, tm (>= 0.6), topicmodels, xtable, knitr, igraph, wordcloud
URL https://quanteda.io
Encoding UTF-8
BugReports https://github.com/quanteda/quanteda/issues
LazyData TRUE
VignetteBuilder knitr
Language en-GB
Collate 'RcppExports.R' 'View.R' 'bootstrap_dfm.R'
  'casechange-functions.R' 'directionchange-functions.R'
  'character-methods.R' 'convert.R' 'corpus-methods-base.R'
  'corpus-methods-quanteda.R' 'corpus-methods-tm.R' 'corpus.R'
'corpus_reshape.R' 'corpus_sample.R' 'corpus_segment.R'
'corpus_subset.R' 'corpus_trim.R' 'corpuszip.R'
'data-deprecated.R' 'data-documentation.R'
'defunct-functions.R' 'dfm-classes.R' 'dfm-methods.R'
'dfm-print.R' 'dfm-subsetting.R' 'dfm.R' 'dfm_compress.R'
'dfm_group.R' 'dfm_lookup.R' 'dfm_replace.R' 'dfm_sample.R'
'dfm_select.R' 'dfm_sort.R' 'dfm_subset.R' 'dfm_trim.R'
'dfm_weight.R' 'dfm_match.R' 'dictionaries.R' 'docnames.R'
'dimnames.R' 'docvars.R' 'dfm-classes.R' 'dfm-methods.R'
'fcm-subsetting.R' 'fcm.R' 'future.R' 'kwic.R' 'nfunctions.R'
'nsrabbare.R' 'nsyllable.R' 'pattern2fixed.R' 'phrases.R'
'quanteda-documentation.R' 'quanteda_options.R'
'readtext-methods.R' 'settings.R' 'spacyr-methods.R'
'stopwords.R' 'textmodel-methods.R' 'textmodel_affinity.R'
'textmodel_ca.R' 'textmodel_lsa.R' 'textmodel_nb.R'
'textmodel_wordfish.R' 'textmodel_wordscores.R'
'textplot_influence.R' 'textplot_keyness.R'
'textplot_network.R' 'textplot_scale1d.R'
'textplot_wordcloud.R' 'textplot_xray.R' 'textstat-methods.R'
'textstat_collocations.R' 'textstat_dist_old.R'
'textstat_frequency.R' 'textstat_keyness.R' 'textstat_lexdiv.R'
'textstat_readability.R' 'textstat_simil.R'
'textstat_simil_old.R' 'textstat_entropy.R' 'tokens.R'
'tokens_compound.R' 'tokens_group.R' 'tokens_lookup.R'
'tokens_ngrams.R' 'tokens_replace.R' 'tokens_segment.R'
'tokens_select.R' 'tokens_subset.R' 'tokens_sample.R'
'tokens_split.R' 'tokens_chunk.R' 'utils.R' 'wordstem.R'
'zzz.R'

RoxygenNote 7.0.1

SystemRequirements C++11

NeedsCompilation yes

Author Kenneth Benoit [cre, aut, cph]
<Kohei Watanabe [aut] (https://orcid.org/0000-0001-6519-5265),
Haiyan Wang [aut] (https://orcid.org/0000-0003-4992-4311),
Paul Nulty [aut] (https://orcid.org/0000-0002-7214-4666),
Adam Obeng [aut] (https://orcid.org/0000-0002-2906-4775),
Stefan Müller [aut] (https://orcid.org/0000-0002-6315-4125),
Akitaka Matsuo [aut] (https://orcid.org/0000-0002-3323-6330),
Jiong Wei Lua [aut],
Patrick O. Perry [aut] (https://orcid.org/0000-0001-7460-127X),
Jouni Kuha [aut] (https://orcid.org/0000-0002-1156-8465),
Benjamin Lauderdale [aut] (https://orcid.org/0000-0003-3090-0969),
William Lowe [aut] (https://orcid.org/0000-0002-1549-6163),
Christian Müller [ctb],
Lori Young [dtc] (Lexicoder Sentiment Dictionary 2015),
Stuart Soroka [dtc] (Lexicoder Sentiment Dictionary 2015),>
R topics documented:

Ian Fellows [cph] (authored wordcloud C source code (modified)),
European Research Council [fnd] (ERC-2011-StG 283794-QUANTESS)

Maintainer Kenneth Benoit <kbenoit@lse.ac.uk>
Repository CRAN
Date/Publication 2019-11-26 06:40:03 UTC

R topics documented:

quanteda-package .......................................................... 5
as.corpus.corpuszip ....................................................... 7
as.dfm ................................................................. 7
as.dictionary .......................................................... 8
as.list.dist ........................................................ 9
as.matrix.dfm ......................................................... 9
as.tokens .............................................................. 10
as.yaml .............................................................. 12
bootstrap_dfm ......................................................... 13
char_tolower .......................................................... 14
convert .............................................................. 14
corpus .............................................................. 16
corpus_reshape ....................................................... 20
corpus_sample ....................................................... 21
corpus_segment ...................................................... 22
corpus_subset ....................................................... 24
corpus_trim ........................................................ 25
data_char_sampletext .................................................. 26
data_char_ukimmig2010 ................................................. 27
data_corpus_dailnoconf1991 .......................................... 28
data_corpus_inaugural ................................................. 29
data_corpus_irishbudget2010 ......................................... 30
data_dfm_lbgexample .................................................... 30
data_dictionary_LSD2015 .............................................. 31
dfm ................................................................. 33
dfm_compress ......................................................... 35
dfm_group .......................................................... 37
dfm_lookup .......................................................... 38
dfm_match ............................................................ 40
dfm_replace .......................................................... 41
dfm_sample .......................................................... 42
dfm_select ........................................................... 43
dfm_sort .............................................................. 45
dfm_subset ........................................................... 46
dfm_tfidf ............................................................. 47
dfm_tolower .......................................................... 49
dfm_trim .............................................................. 50
dfm_weight ........................................................... 52
dictionary ............................................................ 54
R topics documented:

docfreq .......................................................... 56
docnames ......................................................... 58
docvars ............................................................ 59
fcm ................................................................. 60
fcm_sort ......................................................... 62
featfreq .......................................................... 63
featnames ......................................................... 63
head.corpus ....................................................... 64
head.dfm .......................................................... 65
kwie ................................................................. 66
metacorpus ....................................................... 67
metadoc ........................................................... 68
ndoc ................................................................. 69
nscrabble ........................................................ 70
nsentence ......................................................... 71
nsyllable ......................................................... 71
ntoken .............................................................. 73
phrase ............................................................. 74
quanteda_options ............................................... 75
spacyr-methods .................................................. 76
sparsity ........................................................... 77
textmodel_affinity ............................................. 78
textmodel_ca ..................................................... 79
textmodel_lsa .................................................... 80
textmodel_nb ..................................................... 81
textmodel_wordfish ............................................. 84
textplot_influence .............................................. 88
textplot_keyness ............................................... 89
textplot_network ............................................... 90
textplot_scale1d ............................................... 92
textplot_wordcloud ............................................ 94
textplot_xray ................................................... 96
texts ............................................................... 98
textstat_collocations ......................................... 99
textstat_entropy ............................................... 101
textstat_frequency ............................................ 102
textstat_keyness ............................................... 104
textstat_lexdiv ............................................... 105
textstat_readability .......................................... 109
textstat_simil .................................................. 117
tokens ........................................................... 120
tokens_chunk .................................................... 124
tokens_compound ............................................... 125
tokens_lookup ................................................... 126
tokens_ngrams .................................................. 129
tokens_replace .................................................. 130
tokens_sample ................................................... 132
Description

A set of functions for creating and managing text corpora, extracting features from text corpora, and analyzing those features using quantitative methods.

**quanteda** makes it easy to manage texts in the form of a corpus, defined as a collection of texts that includes document-level variables specific to each text, as well as meta-data for documents and for the collection as a whole. **quanteda** includes tools to make it easy and fast to manipulate the texts in a corpus, by performing the most common natural language processing tasks simply and quickly, such as tokenizing, stemming, or forming ngrams. **quanteda**’s functions for tokenizing texts and forming multiple tokenized documents into a document-feature matrix are both extremely fast and extremely simple to use. **quanteda** can segment texts easily by words, paragraphs, sentences, or even user-supplied delimiters and tags.

Built on the text processing functions in the **stringi** package, which is in turn built on C++ implementation of the ICU libraries for Unicode text handling, **quanteda** pays special attention to fast and correct implementation of Unicode and the handling of text in any character set. **quanteda** is built for efficiency and speed, through its design around three infrastructures: the **stringi** package for text processing, the **data.table** package for indexing large documents efficiently, and the **Matrix** package for sparse matrix objects. If you can fit it into memory, **quanteda** will handle it quickly. (And eventually, we will make it possible to process objects even larger than available memory.)

**quanteda** is principally designed to allow users a fast and convenient method to go from a corpus of texts to a selected matrix of documents by features, after defining what the documents and features. The package makes it easy to redefine documents, for instance by splitting them into sentences or paragraphs, or by tags, as well as to group them into larger documents by document variables, or to subset them based on logical conditions or combinations of document variables. The package also implements common NLP feature selection functions, such as removing stopwords and stemming in numerous languages, selecting words found in dictionaries, treating words as equivalent based on a user-defined "thesaurus", and trimming and weighting features based on document frequency, feature frequency, and related measures such as tf-idf.

Once constructed, a **quanteda** document-feature matrix ("dfm") can be easily analyzed using either **quanteda**’s built-in tools for scaling document positions, or used with a number of other text analytic tools, such as: topic models (including converters for direct use with the topicmodels,
LDA, and stm packages) document scaling (using quanteda's own functions for the "wordfish" and "Wordscores" models, direct use with the ca package for correspondence analysis, or scaling with the austin package) machine learning through a variety of other packages that take matrix or matrix-like inputs.

Additional features of quanteda include:

• powerful, flexible tools for working with dictionaries;
• the ability to identify keywords associated with documents or groups of documents;
• the ability to explore texts using key-words-in-context;
• fast computation of a variety of readability indexes;
• fast computation of a variety of lexical diversity measures;
• quick computation of word or document similarities, for clustering or to compute distances for other purposes;
• a comprehensive suite of descriptive statistics on text such as the number of sentences, words, characters, or syllables per document; and
• flexible, easy to use graphical tools to portray many of the analyses available in the package.

Source code and additional information

http://github.com/quanteda/quanteda

Author(s)

Maintainer: Kenneth Benoit <kbenoit@lse.ac.uk> (ORCID) [copyright holder]

Authors:

• Kohei Watanabe <watanabe.kohei@gmail.com> (ORCID)
• Haiyan Wang <whyinsha@yahoo.com> (ORCID)
• Paul Nulty <paul.nulty@gmail.com> (ORCID)
• Adam Obeng <quanteda@binaryeagle.com> (ORCID)
• Stefan Müller <mullers@tcd.ie> (ORCID)
• Akitaka Matsuo <a.matsu@lse.ac.uk> (ORCID)
• Jiong Wei Lua <J.W.Lua@lse.ac.uk>
• Patrick O. Perry <patperry@gmail.com> (ORCID)
• Jouni Kuha <j.kuha@lse.ac.uk> (ORCID)
• Benjamin Lauderdale <b.e.lauderdale@lse.ac.uk> (ORCID)
• William Lowe <wlowe@princeton.edu> (ORCID)

Other contributors:

• Christian Müller <C.Mueller@lse.ac.uk> [contributor]
• Lori Young (Lexicoder Sentiment Dictionary 2015) [data contributor]
• Stuart Soroka (Lexicoder Sentiment Dictionary 2015) [data contributor]
• Ian Fellows <ian@fellstat.com> (authored wordcloud C source code (modified)) [copyright holder]
• European Research Council (ERC-2011-StG 283794-QUANTESS) [funder]
as.corpus.corpuszip

See Also

Useful links:
- https://quanteda.io
- Report bugs at https://github.com/quanteda/quanteda/issues

as.corpus.corpuszip  Coerce a compressed corpus to a standard corpus

Description

Recast a compressed corpus object into a standard (uncompressed) corpus object.

Usage

```r
## S3 method for class 'corpuszip'
as.corpus(x)
```

Arguments

- `x`  
  a compressed `corpus` object

as.dfm  Coercion and checking functions for dfm objects

Description

Convert an eligible input object into a dfm, or check whether an object is a dfm. Current eligible inputs for coercion to a dfm are: matrix, (sparse) Matrix, TermDocumentMatrix, DocumentTermMatrix, data.frame, and other dfm objects.

Usage

```r
as.dfm(x)

is.dfm(x)
```

Arguments

- `x`  
  a candidate object for checking or coercion to dfm

Value

- `as.dfm` converts an input object into a dfm. Row names are used for docnames, and column names for featnames, of the resulting dfm.
- `is.dfm` returns TRUE if and only if its argument is a dfm.
Coercion and checking functions for dictionary objects

Description
Convert a dictionary from a different format into a quanteda dictionary, or check to see if an object is a dictionary.

Usage
as.dictionary(x, format = c("tidytext"), separator = " ", tolower = FALSE)

Arguments
x
a dictionary-like object to be coerced or checked
format
input format for the object to be coerced to a dictionary; current legal values are a data.frame with the fields word and sentiment (as per the tidytext package)
separator
the character in between multi-word dictionary values. This defaults to " ".
tolower
if TRUE, convert all dictionary values to lowercase

Value
as.dictionary returns a quanteda dictionary object. This conversion function differs from the dictionary constructor function in that it converts an existing object rather than creates one from components or from a file.
is.dictionary returns TRUE if an object is a quanteda dictionary.

Examples
## Not run:
data(sentiments, package = "tidytext")
as.dictionary(subset(sentiments, lexicon == "nrc"))
as.dictionary(subset(sentiments, lexicon == "bing"))
# to convert AFINN into polarities - adjust thresholds if desired
datafinn <- subset(sentiments, lexicon == "AFINN")
datafinn[["sentiment"]]
with(datafinn,
    sentiment <- ifelse(score < 0, "negative",
                        ifelse(score > 0, "positive", "netural"))
)
with(datafinn, table(score, sentiment))
as.dictionary(datafinn)
as.list.dist

Coerce a dist object into a list

Description

Coerce a dist matrix into a list of selected target terms and similar terms, in descending order of similarity. Can be used after calling textstat_simil or textstat.dist.

Usage

## S3 method for class 'dist'
as.list(x, sorted = TRUE, n = NULL, ...)

Arguments

x dist class object
sorted sort results in descending order if TRUE
n the top n highest-ranking items will be returned. If n is NULL, return all items.
...

unused

as.matrix.dfm

Coerce a dfm to a matrix or data.frame

Description

Methods for coercing a dfm object to a matrix or data.frame object.

Usage

## S3 method for class 'dfm'
as.matrix(x, ...)

dat <- data.frame(
  word = c("Great", "Horrible"),
  sentiment = c("positive", "negative")
)

as.dictionary(dat)
as.dictionary(dat, tolower = FALSE)

# End(Not run)

is.dictionary(dictionary(list(key1 = c("val1", "val2"), key2 = "val3")))
# [1] TRUE

is.dictionary(list(key1 = c("val1", "val2"), key2 = "val3"))
# [1] FALSE
Arguments
  x  dfm to be coerced
  ...  unused

Examples
  # coercion to matrix
  as.matrix(data_dfm_lbgexample[, 1:10])

Description
  Coercion functions to and from tokens objects, checks for whether an object is a tokens object, and
  functions to combine tokens objects.

Usage
  as.tokens(x, concatenator = "_", ...)

  ## S3 method for class 'list'
  as.tokens(x, concatenator = "_", ...)

  ## S3 method for class 'spacyr_parsed'
  as.tokens(
    x,
    concatenator = "/",
    include_pos = c("none", "pos", "tag"),
    use_lemma = FALSE,
    ...
  )

  ## S3 method for class 'tokens'
  as.list(x, ...)

  ## S3 method for class 'tokens'
  as.character(x, use.names = FALSE, ...)

  is.tokens(x)

  ## S3 method for class 'tokens'
  unlist(x, recursive = FALSE, use.names = TRUE)

  ## S3 method for class 'tokens'
## S3 method for class 'tokens'
c(...)

### Arguments

- **x**: object to be coerced or checked
- **concatenator**: character between multi-word expressions, default is the underscore character. See Details.
- **...**: additional arguments used by specific methods. For `c.tokens`, these are the `tokens` objects to be concatenated.
- **include_pos**: character; whether and which part-of-speech tag to use: "none" do not use any part of speech indicator, "pos" use the pos variable, "tag" use the tag variable. The POS will be added to the token after "concatenator".
- **use_lemma**: logical; if TRUE, use the lemma rather than the raw token
- **use.names**: logical; preserve names if TRUE. For `as.character` and `unlist` only.
- **recursive**: a required argument for `unlist` but inapplicable to `tokens` objects
- **t1**: tokens one to be added
- **t2**: tokens two to be added

### Details

The concatenator is used to automatically generate dictionary values for multi-word expressions in `tokens_lookup` and `dfm_lookup`. The underscore character is commonly used to join elements of multi-word expressions (e.g. "piece_of_cake", "New_York"), but other characters (e.g. whitespace " " or a hyphen ".") can also be used. In those cases, users have to tell the system what is the concatenator in your tokens so that the conversion knows to treat this character as the inter-word delimiter, when reading in the elements that will become the tokens.

### Value

- `as.tokens` returns a quanteda `tokens` object.
- `as.list` returns a simple list of characters from a `tokens` object.
- `as.character` returns a character vector from a `tokens` object.
- `is.tokens` returns TRUE if the object is of class tokens, FALSE otherwise.
- `unlist` returns a simple vector of characters from a `tokens` object.
- `c(...)` and `+` return a tokens object whose documents have been added as a single sequence of documents.

### Examples

```r
# create tokens object from list of characters with custom concatenator
dict <- dictionary(list(country = "United States",
               sea = c("Atlantic Ocean", "Pacific Ocean")))
```
litis <- list(c("The", "United-States", "has", "the", "Atlantic-Ocean", "and", "the", "Pacific-Ocean", "."))
toks <- as.tokens(litis, concatenator = "-")
tokens_lookup(toks, dict)

# combining tokens
toks1 <- tokens(c(doc1 = "a b c d e", doc2 = "f g h"))
toks2 <- tokens(c(doc3 = "1 2 3"))
toks1 + toks2
c(toks1, toks2)

---

**as.yaml**

Convert quanteda dictionary objects to the YAML format

**Description**

Converts a quanteda dictionary object constructed by the dictionary function into the YAML format. The YAML files can be edited in text editors and imported into quanteda again.

**Usage**

as.yaml(x)

**Arguments**

- **x**
  a dictionary object

**Value**

as.yaml a dictionary in the YAML format, as a character object

**Examples**

```r
## Not run:
dict <- dictionary(list(one = c("a b", "c*"), two = c("x", "y", "z??")))
cat(yaml <- as.yaml(dict))
cat(yaml, file = (yamlfile <- paste0(tempfile(), ".yml")))
dictionary(file = yamlfile)
## End(Not run)
```
**bootstrap_dfm**

**Bootstrap a dfm**

**Description**
Create an array of resampled dfms.

**Usage**

```r
bootstrap_dfm(x, n = 10, ..., verbose = quanteda_options("verbose"))
```

**Arguments**

- `x`: a character or corpus object
- `n`: number of resamples
- `...`: additional arguments passed to `dfm`
- `verbose`: if TRUE print status messages

**Details**

Function produces multiple, resampled dfm objects, based on resampling sentences (with replacement) from each document, recombining these into new "documents" and computing a dfm for each. Resampling of sentences is done strictly within document, so that every resampled document will contain at least some of its original tokens.

**Value**

A named list of dfm objects, where the first, dfm_0, is the dfm from the original texts, and subsequent elements are the sentence-resampled dfms.

**Author(s)**

Kenneth Benoit

**Examples**

```r
# bootstrapping from the original text
set.seed(10)
txt <- c(textone = "This is a sentence. Another sentence. Yet another.",
         texttwo = "Premiere phrase. Deuxieme phrase.")
bootstrap_dfm(txt, n = 3, verbose = TRUE)
```
**char_tolower**  
*Convert the case of character objects*

**Description**

`char_tolower` and `char_toupper` are replacements for `tolower` and `toupper` based on the `stringi` package. The `stringi` functions for case conversion are superior to the `base` functions because they correctly handle case conversion for Unicode. In addition, the `*_tolower` functions provide an option for preserving acronyms.

**Usage**

```r
char_tolower(x, keep_acronyms = FALSE)
char_toupper(x)
```

**Arguments**

- `x` the input object whose character/tokens/feature elements will be case-converted
- `keep_acronyms` logical; if TRUE, do not lowercase any all-uppercase words (applies only to `*_tolower` functions)

**Examples**

```r
taxt1 <- c(txt1 = "b A A", txt2 = "C C a b B")
char_tolower(txt1)
char_toupper(txt1)

# with acronym preservation
taxt2 <- c(txt1 = "England and France are members of NATO and UNESCO
text2 = "NASA sent a rocket into space.")
char_tolower(txt2)
char_tolower(txt2, keep_acronyms = TRUE)
char_toupper(txt2)
```

**convert**  
*Convert a dfm to a non-quanteda format*

**Description**

Convert a quanteda `dfm` object to a format useable by other text analysis packages. The general function `convert` provides easy conversion from a `dfm` to the document-term representations used in all other text analysis packages for which conversions are defined.
Usage

```r
convert(
  x,
  to = c("lda", "tm", "stm", "austin", "topicmodels", "lsa", "matrix", "data.frame", "tripletlist"),
  docvars = NULL,
  omit_empty = TRUE
)
```

Arguments

- `x` - a dfm to be converted
- `to` - target conversion format, consisting of the name of the package into whose document-term matrix representation the dfm will be converted:
  - "lda" a list with components "documents" and "vocab" as needed by the function `lda.collapsed.gibbs.sampler` from the `lda` package
  - "tm" a `DocumentTermMatrix` from the `tm` package
  - "stm" the format for the `stm` package
  - "austin" the wfm format from the `austin` package
  - "topicmodels" the "dtm" format as used by the `topicmodels` package
  - "lsa" the "textmatrix" format as used by the `lsa` package
  - "data.frame" a data.frame where each feature is a variable
  - "tripletlist" a named "triplet" format list consisting of document, feature, and frequency
- `docvars` - optional data.frame of document variables used as the meta information in conversion to the `stm` package format. This aids in selecting the document variables only corresponding to the documents with non-zero counts. Only affects the "stm" format.
- `omit_empty` - logical; if TRUE, omit empty documents and features from the converted dfm. This is required for some formats (such as STM) that do not accept empty documents. Only used when `to` = "lda" or `to` = "topicmodels". For `to` = "stm" format, 'omit_empty' is always TRUE.

Value

A converted object determined by the value of `to` (see above). See conversion target package documentation for more detailed descriptions of the return formats.

Examples

```r
corp <- corpus_subset(data_corpus_inaugural, Year > 1970)
dfmat1 <- dfm(corp)

# austin's wfm format
identical(dim(dfmat1), dim(convert(dfmat1, to = "austin")))

# stm package format
```
corpus <- convert(dfmat1, to = "stm")
str(stmmat)

#' # triplet
tripletmat <- convert(dfmat1, to = "tripletlist")
str(tripletmat)

# illustrate what happens with zero-length documents
dfmat2 <- dfm(c(punctOnly = "!!!", corp[-1]))
rowSums(dfmat2)
str(convert(dfmat2, to = "stm", docvars = docvars(corp)))

## Not run:
# tm's DocumentTermMatrix format
tmdfm <- convert(dfmat1, to = "tm")
str(tmdfm)

# topicmodels package format
str(convert(dfmat1, to = "topicmodels"))

# lda package format
str(convert(dfmat1, to = "lda"))

## End(Not run)

corpus

**Construct a corpus object**

**Description**

Creates a corpus object from available sources. The currently available sources are:

- a character vector, consisting of one document per element; if the elements are named, these names will be used as document names.
- a data.frame (or a tibble tbl_df), whose default document id is a variable identified by docid_field; the text of the document is a variable identified by text_field; and other variables are imported as document-level meta-data. This matches the format of data.frames constructed by the the readtext package.
- a kwic object constructed by kwic.
- a tm VCorpus or SimpleCorpus class object, with the fixed metadata fields imported as docvars and corpus-level metadata imported as metacorpus information.
- a corpus object.

**Usage**

corpus(x, ...)

## S3 method for class 'corpus'
corpus(
  x,
  docnames = quanteda::docnames(x),
  docvars = quanteda::docvars(x),
  metacorpus = quanteda::metacorpus(x),
  compress = FALSE,
  ...
)

## S3 method for class 'character'
corpus(
  x,
  docnames = NULL,
  docvars = NULL,
  metacorpus = NULL,
  compress = FALSE,
  ...
)

## S3 method for class 'data.frame'
corpus(
  x,
  docid_field = "doc_id",
  text_field = "text",
  metacorpus = NULL,
  compress = FALSE,
  ...
)

## S3 method for class 'kwic'
corpus(x, split_context = TRUE, extract_keyword = TRUE, ...)

## S3 method for class 'Corpus'
corpus(x, metacorpus = NULL, compress = FALSE, ...)

### Arguments

- **x**: a valid corpus source object
- **...**: not used directly
- **docnames**: Names to be assigned to the texts. Defaults to the names of the character vector (if any); doc_id for a data.frame; the document names in a tm corpus; or a vector of user-supplied labels equal in length to the number of documents. If none of these are round, then "text1", "text2", etc. are assigned automatically.
- **docvars**: a data.frame of document-level variables associated with each text
- **metacorpus**: a named list containing additional (character) information to be added to the corpus as corpus-level metadata. Special fields recognized in the `summary.corpus` are:
- source a description of the source of the texts, used for referencing;
- citation information on how to cite the corpus; and
- notes any additional information about who created the text, warnings, to do lists, etc.

compress  logical; if TRUE, compress the texts in memory using gzip compression. This significantly reduces the size of the corpus in memory, but will slow down operations that require the texts to be extracted.

docid_field  optional column index of a document identifier; defaults to "doc_id", but if this is not found, then will use the rownames of the data.frame; if the rownames are not set, it will use the default sequence based on (quanteda_options("base_docname")).

text_field  the character name or numeric index of the source data.frame indicating the variable to be read in as text, which must be a character vector. All other variables in the data.frame will be imported as docvars. This argument is only used for data.frame objects (including those created by readtext).

split_context  logical; if TRUE, split each kwic row into two "documents", one for "pre" and one for "post", with this designation saved in a new docvar context and with the new number of documents therefore being twice the number of rows in the kwic.

extract_keyword  logical; if TRUE, save the keyword matching pattern as a new docvar keyword

Details
The texts and document variables of corpus objects can also be accessed using index notation. Indexing a corpus object as a vector will return its text, equivalent to texts(x). Note that this is not the same as subsetting the entire corpus – this should be done using the subset method for a corpus.

Indexing a corpus using two indexes (integers or column names) will return the document variables, equivalent to docvars(x). It is also possible to access, create, or replace docvars using list notation, e.g.

myCorpus["newSerialDocvar"] <- paste0("tag", 1:ndoc(myCorpus)).

For details, see corpus-class.

Value
A corpus-class class object containing the original texts, document-level variables, document-level metadata, corpus-level metadata, and default settings for subsequent processing of the corpus.

A warning on accessing corpus elements
A corpus currently consists of an S3 specially classed list of elements, but you should not access these elements directly. Use the extractor and replacement functions instead, or else your code is not only going to be uglier, but also likely to break should the internal structure of a corpus object change (as it inevitably will as we continue to develop the package, including moving corpus objects to the S4 class system).
corpus

Author(s)
Kenneth Benoit and Paul Nulty

See Also
corpus-class, docvars, metadoc, metacorpus, settings, texts, ndoc, docnames

Examples

# create a corpus from texts
corpus(data_char_ukimmig2010)

# create a corpus from texts and assign meta-data and document variables
summary(corpus(data_char_ukimmig2010,
    docvars = data.frame(party = names(data_char_ukimmig2010))), 5)

corpus(texts(data_corpus_irishbudget2010))

# import a tm VCorpus
if (requireNamespace("tm", quietly = TRUE)) {

data(crude, package = "tm") # load in a tm example VCorpus
mytmCorpus <- corpus(crude)
summary(mytmCorpus, showmeta=TRUE)

data(acq, package = "tm")
summary(corpus(acq), 5, showmeta=TRUE)

    tmCorp <- tm::VCorpus(tm::VectorSource(data_char_ukimmig2010))
    quantCorp <- corpus(tmCorp)
    summary(quantCorp)
}

# construct a corpus from a data.frame
dat <- data.frame(letter_factor = factor(rep(letters[1:3], each = 2)),
    some_ints = 1L:6L,
    some_text = paste0("This is text number ", 1:6, "."),
    stringsAsFactors = FALSE,
    row.names = paste0("fromDf_", 1:6))
dat
summary(corpus(dat, text_field = "some_text",
    metacorpus = list(source = "From a data.frame called mydf.")))

# construct a corpus from a kwic object
mykwic <- kwic(data_corpus_inaugural, "southern")
summary(corpus(mykwic))

# from a kwic
kw <- kwic(data_char_samptext, "econom*")
summary(corpus(kw))
summary(corpus(kw, split_context = FALSE))
texts(corpus(kw, split_context = FALSE))
Recast the document units of a corpus

description
For a corpus, reshape (or recast) the documents to a different level of aggregation. Units of aggregation can be defined as documents, paragraphs, or sentences. Because the corpus object records its current "units" status, it is possible to move from recast units back to original units, for example from documents, to sentences, and then back to documents (possibly after modifying the sentences).

usage

```r
corpus_reshape(
x,
to = c("sentences", "paragraphs", "documents"),
use_docvars = TRUE,
...
)
```

arguments

- `x`: corpus whose document units will be reshaped
- `to`: new document units in which the corpus will be recast
- `use_docvars`: if TRUE, repeat the docvar values for each segmented text; if FALSE, drop the docvars in the segmented corpus. Dropping the docvars might be useful in order to conserve space or if these are not desired for the segmented corpus.
- `...`: additional arguments passed to `tokens`, since the syntactic segmenter uses this function

value

A corpus object with the documents defined as the new units, including document-level meta-data identifying the original documents.

examples

```r
# simple example
corp1 <- corpus(c(textone = "This is a sentence. Another sentence. Yet another."),
                 texttwo = "Premiere phrase. Deuxieme phrase."),
                 docvars = data.frame(country=c("UK", "USA"), year=c(1990, 2000)),
                 metacorpus = list(notes = "Example showing how corpus_reshape() works."))
summary(corp1)
summary(corpus_reshape(corp1, to = "sentences"), showmeta = TRUE)

# example with inaugural corpus speeches
(corp2 <- corpus_subset(data_corpus_inaugural, Year>2004))
corp2para <- corpus_reshape(corp2, to = "paragraphs")
```
corpus_sample

Randomly sample documents from a corpus

description

Take a random sample of documents of the specified size from a corpus, with or without replacement. Works just as sample works for the documents and their associated document-level variables.

Usage

corpus_sample(x, size = ndoc(x), replace = FALSE, prob = NULL, by = NULL)

Arguments

x
a corpus object whose documents will be sampled

size
a positive number, the number of documents to select; when used with groups, the number to select from each group or a vector equal in length to the number of groups defining the samples to be chosen in each group category. By defining a size larger than the number of documents, it is possible to oversample groups.

replace
Should sampling be with replacement?

prob
A vector of probability weights for obtaining the elements of the vector being sampled. May not be applied when by is used.

by
a grouping variable for sampling. Useful for resampling sub-document units such as sentences, for instance by specifying by = "document"

value

A corpus object with number of documents equal to size, drawn from the corpus x. The returned corpus object will contain all of the meta-data of the original corpus, and the same document variables for the documents selected.

Examples

set.seed(2000)
# sampling from a corpus
summary(corpus_sample(data_corpus_inaugural, 5))
summary(corpus_sample(data_corpus_inaugural, 10, replace = TRUE))

# sampling sentences within document
corp <- corpus(c(one = "Sentence one. Sentence two. Third sentence.",
                two = "First sentence, doc2. Second sentence, doc2."
))
corpsent <- corpus_reshape(corp, to = "sentences")
texts(corpsent)
texts(corpus_sample(corpsent, replace = TRUE, by = "document"))
corpus_segment Segment texts on a pattern match

Description
Segment corpus text(s) or a character vector, splitting on a pattern match. This is useful for breaking the texts into smaller documents based on a regular pattern (such as a speaker identifier in a transcript) or a user-supplied annotation.

Usage
corpus_segment(
  x,
  pattern = "##*",
  valuetype = c("glob", "regex", "fixed"),
  case_insensitive = TRUE,
  extract_pattern = TRUE,
  pattern_position = c("before", "after"),
  use_docvars = TRUE
)

char_segment(
  x,
  pattern = "##*",
  valuetype = c("glob", "regex", "fixed"),
  case_insensitive = TRUE,
  remove_pattern = TRUE,
  pattern_position = c("before", "after")
)

Arguments
x character or corpus object whose texts will be segmented
pattern a character vector, list of character vectors, dictionary, or collocations object. See pattern for details.
valuetype the type of pattern matching: "glob" for "glob"-style wildcard expressions; "regex" for regular expressions; or "fixed" for exact matching. See valuetype for details.
case_insensitive ignore case when matching, if TRUE
extract_pattern extracts matched patterns from the texts and save in docvars if TRUE
pattern_position either "before" or "after", depending on whether the pattern precedes the text (as with a user-supplied tag, such as ##INTRO in the examples below) or follows the text (as with punctuation delimiters)
corpus_segment

use_docvars if TRUE, repeat the docvar values for each segmented text; if FALSE, drop the docvars in the segmented corpus. Dropping the docvars might be useful in order to conserve space or if these are not desired for the segmented corpus.

remove_pattern removes matched patterns from the texts if TRUE

Details

For segmentation into syntactic units defined by the locale (such as sentences), use corpus_reshape instead. In cases where more fine-grained segmentation is needed, such as that based on commas or semi-colons (phrase delimiters within a sentence), corpus_segment offers greater user control than corpus_reshape.

Value

corpus_segment returns a corpus of segmented texts

char_segment returns a character vector of segmented texts

Boundaries and segmentation explained

The pattern acts as a boundary delimiter that defines the segmentation points for splitting a text into new "document" units. Boundaries are always defined as the pattern matches, plus the end and beginnings of each document. The new "documents" that are created following the segmentation will then be the texts found between boundaries.

The pattern itself will be saved as a new document variable named pattern. This is most useful when segmenting a text according to tags such as names in a transcript, section titles, or user-supplied annotations. If the beginning of the file precedes a pattern match, then the extracted text will have a NA for the extracted pattern document variable (or when pattern_position = "after", this will be true for the text split between the last pattern match and the end of the document).

To extract syntactically defined sub-document units such as sentences and paragraphs, use corpus_reshape instead.

Using patterns

One of the most common uses for corpus_segment is to partition a corpus into sub-documents using tags. The default pattern value is designed for a user-annotated tag that is a term beginning with double "hash" signs, followed by a whitespace, for instance as #INTRODUCTION The text.

Glob and fixed pattern types use a whitespace character to signal the end of the pattern.

For more advanced pattern matches that could include whitespace or newlines, a regex pattern type can be used, for instance a text such as

Mr. Smith: Text
Mrs. Jones: More text

could have as pattern = \b[A-Z].+\s[A-Z][a-z]+:, which would catch the title, the name, and the colon.

For custom boundary delimitation using punctuation characters that come at the end of a clause or sentence (such as , and .), these can be specified manually and pattern_position set to "after". To keep the punctuation characters in the text (as with sentence segmentation), set
extract_pattern = FALSE. (With most tag applications, users will want to remove the patterns from the text, as they are annotations rather than parts of the text itself.)

See Also
corpus_reshape, for segmenting texts into pre-defined syntactic units such as sentences, paragraphs, or fixed-length chunks

Examples

## segmenting a corpus

# segmenting a corpus using tags
corp1 <- corpus(c("##INTRO This is the introduction.
##DOC1 This is the first document. Second sentence in Doc 1.
","##INTRO Document ##NUMBER Two starts before ##NUMBER Three."))
corpseg1 <- corpus_segment(corp1, pattern = "##*"
cbind(texts(corpseg1), docvars(corpseg1), metadoc(corpseg1))

# segmenting a transcript based on speaker identifiers
corp2 <- corpus("Mr. Smith: Text.
Mrs. Jones: More text.
Mr. Smith: I'm speaking, again.")
corpseg2 <- corpus_segment(corp2, pattern = \b[A-Z].+\s[A-Z][a-z]+:\",
valuetype = "regex")
cbind(texts(corpseg2), docvars(corpseg2), metadoc(corpseg2))

# segmenting a corpus using crude end-of-sentence segmentation
corpseg3 <- corpus_segment(corp1, pattern = ".", valuetype = "fixed",
pattern_position = "after", extract_pattern = FALSE)
cbind(texts(corpseg3), docvars(corpseg3), metadoc(corpseg3))

## segmenting a character vector

# segment into paragraphs and removing the "- " bullet points
cat(data_char_ukimmig2010[4])
char_segment(data_char_ukimmig2010[4],
pattern = "\n\n\n\n\n\n\n(0,1)", valuetype = "regex",
remove_pattern = TRUE)

# segment a text into clauses
txt <- c(d1 = "This, is a sentence? You: come here.", d2 = "Yes, yes okay.")
char_segment(txt, pattern = \p{P}", valuetype = "regex",
pattern_position = "after", remove_pattern = FALSE)
corpus_subset

Description

Returns subsets of a corpus that meet certain conditions, including direct logical operations on docvars (document-level variables). corpus_subset functions identically to subset.data.frame, using non-standard evaluation to evaluate conditions based on the docvars in the corpus.

Usage

corpus_subset(x, subset, select, ...)

Arguments

x corpus object to be subsetted
subset logical expression indicating the documents to keep: missing values are taken as false
select expression, indicating the docvars to keep
... not used

Value
corpus object, with a subset of documents (and docvars) selected according to arguments

See Also

subset.data.frame

Examples

summary(corpus_subset(data_corpus_inaugural, Year > 1980))
summary(corpus_subset(data_corpus_inaugural, Year > 1930 & President == "Roosevelt", select = Year))

corpus_trim

Remove sentences based on their token lengths or a pattern match

Description

Removes sentences from a corpus or a character vector shorter than a specified length.

Usage

corpus_trim(
  x,
  what = c("sentences", "paragraphs", "documents"),
  min_ntoken = 1,
  max_ntoken = NULL,
  exclude_pattern = NULL
)
char_trim(
  x,
  what = c("sentences", "paragraphs", "documents"),
  min_ntoken = 1,
  max_ntoken = NULL,
  exclude_pattern = NULL
)

Arguments

x corpus or character object whose sentences will be selected.
what units of trimming, "sentences" or "paragraphs", or "documents"
min_ntoken, max_ntoken minimum and maximum lengths in word tokens (excluding punctuation)
exclude_pattern a string regular expression whose match (at the sentence level) will be used to exclude sentences

Value

a corpus or character vector equal in length to the input. If the input was a corpus, then the all docvars and metadata are preserved. For documents whose sentences have been removed entirely, a null string ("" ) will be returned.

Examples

txt <- c("PAGE 1. This is a single sentence. Short sentence. Three word sentence.",
"PAGE 2. Very short! Shorter.",
"Very long sentence, with multiple parts, separated by commas. PAGE 3.")
corp <- corpus(txt, docvars = data.frame(serial = 1:3))
texts(corp)

# exclude sentences shorter than 3 tokens
texts(corpus_trim(corp, min_ntoken = 3))
# exclude sentences that start with "PAGE <digit(s)>"
texts(corpus_trim(corp, exclude_pattern = "^PAGE \d+"))

# trimming character objects
char_trim(txt, "sentences", min_ntoken = 3)
char_trim(txt, "sentences", exclude_pattern = "sentence\.")

---

**data_char_sampletext**  
A paragraph of text for testing various text-based functions

**Description**

This is a long paragraph (2,914 characters) of text taken from a debate on Joe Higgins, delivered December 8, 2011.
data_char_UKimmig2010

Usage

data_char_sampletext

Format

calendar vector with one element

Source


Examples

tokens(data_char_sampletext, remove_punct = TRUE)

data_char_UKimmig2010  Immigration-related sections of 2010 UK party manifestos

Description

Extracts from the election manifestos of 9 UK political parties from 2010, related to immigration or asylum-seekers.

Usage

data_char_UKimmig2010

Format

A named character vector of plain ASCII texts

Examples

data_corpus_UKimmig2010 <-
corpus(data_char_UKimmig2010, 
docvars = data.frame(party = names(data_char_UKimmig2010)))
metadoc(data_corpus_UKimmig2010, "language") <- "english"
summary(data_corpus_UKimmig2010, showmeta = TRUE)
data_corpus_dailnoconf1991

*Confidence debate from 1991 Irish Parliament*

**Description**

Texts of speeches from a no-confidence motion debated in the Irish Dáil from 16-18 October 1991 over the future of the Fianna Fail-Progressive Democrat coalition. (See Laver and Benoit 2002 for details.)

**Usage**

data_corpus_dailnoconf1991

**Format**

data_corpus_dailnoconf1991 is a corpus with 58 texts, including docvars for name, party, and position.

**Source**


**References**


**Examples**

```r
## Not run:
data_dfm_dailnoconf1991 <- dfm(data_corpus_dailnoconf1991, remove_punct = TRUE)
tmod <- textmodel_affinity(data_dfm_dailnoconf1991,
c(“Govt”, “Opp”, “Opp”, rep(NA, 55)))
pred <- predict(tmod)
dat <-
data.frame(party = as.character(docvars(data_corpus_dailnoconf1991, “party”)),
  govt = coef(pred)[, “Govt”],
  position = as.character(docvars(data_corpus_dailnoconf1991, “position”)),
  stringsAsFactors = FALSE)
bymedian <- with(dat, reorder(paste(party, position), govt, median))
par(mar = c(5, 6, 4, 2)+.1)
boxplot(govt ~ bymedian, data = dat,
  horizontal = TRUE, las = 1,
  xlab = “Degree of support for government”) 
abline(h = 7.5, col = “red”, lty = “dashed”)
```
Description

US presidential inaugural address texts, and metadata (for the corpus), from 1789 to present.

Usage

data_corpus_inaugural

Format

a corpus object with the following docvars:

- Year a four-digit integer year
- President character; President’s last name
- FirstName character; President’s first name (and possibly middle initial)

Details

data_corpus_inaugural is the quanteda-package corpus object of US presidents’ inaugural addresses since 1789. Document variables contain the year of the address and the last name of the president.

Source


Examples

# some operations on the inaugural corpus
summary(data_corpus_inaugural)
head(docvars(data_corpus_inaugural), 10)
**data_corpus_irishbudget2010**  
_Irish budget speeches from 2010_

**Description**  
Speeches and document-level variables from the debate over the Irish budget of 2010.

**Usage**  
`data_corpus_irishbudget2010`

**Format**  
The corpus object for the 2010 budget speeches, with document-level variables for year, debate, serial number, first and last name of the speaker, and the speaker's party.

**Details**  
At the time of the debate, Fianna Fáil (FF) and the Greens formed the government coalition, while Fine Gael (FG), Labour (LAB), and Sinn Féin (SF) were in opposition.

**Source**  

**References**  

**Examples**  
`summary(data_corpus_irishbudget2010)`

---

**data_dfm_lbgexample**  
_dfm from data in Table 1 of Laver, Benoit, and Garry (2003)_

**Description**  
Constructed example data to demonstrate the Wordscores algorithm, from Laver Benoit and Garry (2003), Table 1.

**Usage**  
`data_dfm_lbgexample`
**data_dictionary_LSD2015**

**Format**

A `dfm` object with 6 documents and 37 features.

**Details**

This is the example word count data from Laver, Benoit and Garry’s (2003) Table 1. Documents R1 to R5 are assumed to have known positions: -1.5, -0.75, 0, 0.75, 1.5. Document V1 is assumed unknown, and will have a raw text score of approximately -0.45 when computed as per LBG (2003).

**References**


---

**data_dictionary_LSD2015**

*Lexicoder Sentiment Dictionary (2015)*

**Description**

The 2015 Lexicoder Sentiment Dictionary in *quanteda* dictionary format.

**Usage**

data_dictionary_LSD2015

**Format**

A dictionary of four keys containing glob-style pattern matches.

- `negative` 2,858 word patterns indicating negative sentiment
- `positive` 1,709 word patterns indicating positive sentiment
- `neg_positive` 1,721 word patterns indicating a positive word preceded by a negation (used to convey negative sentiment)
- `neg_negative` 2,860 word patterns indicating a negative word preceded by a negation (used to convey positive sentiment)

**Details**

The dictionary consists of 2,858 "negative" sentiment words and 1,709 "positive" sentiment words. A further set of 2,860 and 1,721 negations of negative and positive words, respectively, is also included. While many users will find the non-negation sentiment forms of the LSD adequate for sentiment analysis, Young and Soroka (2012) did find a small, but non-negligible increase in performance when accounting for negations. Users wishing to test this or include the negations are encouraged to subtract negated positive words from the count of positive words, and subtract the negated negative words from the negative count.
Young and Soroka (2012) also suggest the use of a pre-processing script to remove specific cases of some words (i.e., "good bye", or "nobody better", which should not be counted as positive). Pre-processing scripts are available at http://lexicoder.com.

License and Conditions

The LSD is available for non-commercial academic purposes only. By using data_dictionary_LSD2015, you accept these terms.

Please cite the references below when using the dictionary.

References


Examples

# simple example

txt <- "This aggressive policy will not win friends."

tokens_lookup(tokens(txt), dictionary = data_dictionary_LSD2015, exclusive = FALSE)
## tokens from 1 document.
## text1 :
## [1] "This" "NEGATIVE" "policy" "will" "NEG_POSITIVE" "POSITIVE" "POSITIVE" "."

# notice that double-counting of negated and non-negated terms is avoided
# when using nested_scope = "dictionary"

tokens_lookup(tokens(txt), dictionary = data_dictionary_LSD2015,
    exclusive = FALSE, nested_scope = "dictionary")
## tokens from 1 document.
## text1 :
## [1] "This" "NEGATIVE" "policy" "will" "NEG_POSITIVE" "POSITIVE."

# on larger examples - notice that few negations are used

dfm(data_char_ukimmig2010, dictionary = data_dictionary_LSD2015)
kwic(data_char_ukimmig2010, "not")

# compound neg_negative and neg_positive tokens before creating a dfm object

toks <- tokens_compound(tokens(txt), data_dictionary_LSD2015)

dfm_lookup(dfm(toks), data_dictionary_LSD2015)
Create a document-feature matrix

Description

Construct a sparse document-feature matrix, from a character, corpus, tokens, or even other dfm object.

Usage

```r
dfm(x, 
  tolower = TRUE, 
  stem = FALSE, 
  select = NULL, 
  remove = NULL, 
  dictionary = NULL, 
  thesaurus = NULL, 
  valuetype = c("glob", "regex", "fixed"), 
  groups = NULL, 
  verbose = quanteda_options("verbose"), 
  ...)
```

Arguments

- `x` character, corpus, tokens, or dfm object
- `tolower` convert all features to lowercase
- `stem` if TRUE, stem words
- `select` a pattern of user-supplied features to keep, while excluding all others. This can be used in lieu of a dictionary if there are only specific features that a user wishes to keep. To extract only Twitter usernames, for example, set `select = "@*"` and make sure that `remove_twitter = FALSE` as an additional argument passed to `tokens`. Note: `select = "^@\w+\b"` would be the regular expression version of this matching pattern. The pattern matching type will be set by `valuetype`. See also `tokens_remove`.
- `remove` a pattern of user-supplied features to ignore, such as "stop words". To access one possible list (from any list you wish), use `stopwords()`. The pattern matching type will be set by `valuetype`. See also `tokens_select`. For behaviour of `remove` with ngrams > 1, see Details.
- `dictionary` a dictionary object to apply to the tokens when creating the dfm
- `thesaurus` a dictionary object that will be applied as if `exclusive = FALSE`. See also `tokens_lookup`. For more fine-grained control over this and other aspects of converting features into dictionary/thesaurus keys from pattern matches to values, consider creating the dfm first, and then applying `dfm_lookup` separately, or using `tokens Lookup` on the tokenized text before calling `dfm`.


valuetype  the type of pattern matching: "glob" for "glob"-style wildcard expressions; "regex" for regular expressions; or "fixed" for exact matching. See valuetype for details.

groups either: a character vector containing the names of document variables to be used for grouping; or a factor or object that can be coerced into a factor equal in length or rows to the number of documents. See groups for details.

verbose display messages if TRUE

... additional arguments passed to tokens; not used when x is a dfm

Details

The default behaviour for remove/select when constructing ngrams using dfm(x, ngrams > 1) is to remove/select any ngram constructed from a matching feature. If you wish to remove these before constructing ngrams, you will need to first tokenize the texts with ngrams, then remove the features to be ignored, and then construct the dfm using this modified tokenization object. See the code examples for an illustration.

To select on and match the features of a another dfm, x must also be a dfm.

Value

a dfm-class object

Note

When x is a dfm, groups provides a convenient and fast method of combining and refactoring the documents of the dfm according to the groups.

See Also

dfm_select, dfm-class

Examples

## for a corpus
dfmat1 <- corpus_subset(data_corpus_inaugural, Year > 1980)
dfm(dfmat1)
dfm(dfmat1, tolower = FALSE)

# grouping documents by docvars in a corpus
dfm(dfmat1, groups = "President", verbose = TRUE)

# with English stopwords and stemming
dfm(dfmat1, remove = stopwords("english"), stem = TRUE, verbose = TRUE)
# works for both words in ngrams too
dfm("Banking industry", stem = TRUE, ngrams = 2)

# with dictionaries
dfmat2 <- corpus_subset(data_corpus_inaugural, Year > 1900)
dict <- dictionary(list(christmas = c("Christmas", "Santa", "holiday"),
opposition = c("Opposition", "reject", "notincorpus"),

...
taxing = "taxing",
taxation = "taxation",
taxregex = "tax*",
country = "states")
dfm(dfm2, dictionary = dict)

# removing stopwords
txt <- "The quick brown fox named Seamus jumps over the lazy dog also named Seamus, with the newspaper from a boy named Seamus, in his mouth."
corp <- corpus(txt)
# note: "also" is not in the default stopwords("english")
featnames(dfm(corp, select = stopwords("english")))
# for ngrams
featnames(dfm(corp, ngrams = 2, select = stopwords("english"), remove_punct = TRUE))
featnames(dfm(corp, ngrams = 1:2, select = stopwords("english"), remove_punct = TRUE))

# removing stopwords before constructing ngrams
toks1 <- tokens(char_tolower(txt), remove_punct = TRUE)
toks2 <- tokens_remove(toks1, stopwords("english"))
toks3 <- tokens_ngrams(toks2, 2)
featnames(dfm(toks3))

# keep only certain words
dfm(corp, select = "*s") # keep only words ending in "s"
dfm(corp, select = "$", valuetype = "regex")

# testing Twitter functions
txttweets <- c("My homie @justinbieber #justinbieber shopping in #LA yesterday #beliebers",
"2all the ha8ers including my bro #justinbieber #emabiggestfansjustinbieber",
"Justin Bieber #justinbieber #belieber #fetusjustin #EMABiggestFansJustinBieber")
dfm(txttweets, select = "#*", remove_twitter = FALSE) # keep only hashtags
dfm(txttweets, select = "^#.*$", valuetype = "regex", remove_twitter = FALSE)

# for a dfm
dfm3 <- dfm(data_corpus_irishbudget2010)
dfm4 <- dfm(dfm3,
groups = ifelse(doccvars(data_corpus_irishbudget2010, "party") %in% c("FF", "Green"),
"Govt", "Opposition"),
tolower = FALSE, verbose = TRUE)

---

**dfm_compress**  
Recombine a dfm or fcm by combining identical dimension elements

**Description**

"Compresses" or groups a dfm or fcm whose dimension names are the same, for either documents or features. This may happen, for instance, if features are made equivalent through application of a thesaurus. It could also be needed after a cbind.dfm or rbind.dfm operation. In most cases, you
will not need to call `dfm_compress`, since it is called automatically by functions that change the dimensions of the dfm, e.g. `dfm_tolower`.

Usage

```r
dfm_compress(x, margin = c("both", "documents", "features"))
```

```r
fcm_compress(x)
```

Arguments

- `x` input object, a dfm or fcm
- `margin` character indicating on which margin to compress a dfm, either "documents", "features", or "both" (default). For fcm objects, "documents" has no effect.
- `...` additional arguments passed from generic to specific methods

Value

`dfm_compress` returns a dfm whose dimensions have been recombined by summing the cells across identical dimension names (docnames or featnames). The docvars will be preserved for combining by features but not when documents are combined.

`fcm_compress` returns an fcm whose features have been recombined by combining counts of identical features, summing their counts.

Note

`fcm_compress` works only when the fcm was created with a document context.

Examples

```r
# dfm_compress examples
dfmat <- rbind(dfm(c("b A A", "C C a b B"), tolower = FALSE),
                dfm("A C C C C C", tolower = FALSE))
columns(dfmat) <- char_tolower(featnames(dfmat))
dfmat
dfm_compress(dfmat, margin = "documents")
dfm_compress(dfmat, margin = "features")
dfm_compress(dfmat)

# no effect if no compression needed
dfmatsubset <- dfm(data_corpus_inaugural[1:5])
dim(dfmatsubset)
dim(dfm_compress(dfmatsubset))

# compress an fcm
fcmat1 <- fcm(tokens("A D a C E a d F e B A C E D"),
               context = "window", window = 3)
## this will produce an error:
# fcm_compress(fcmat1)
```
txt <- c("The fox JUMPED over the dog.",
        "The dog jumped over the fox.")
toks <- tokens(txt, remove_punct = TRUE)
fcm2 <- fcm(toks, context = "document")
colnames(fcm2) <- rownames(fcm2) <- tolower(colnames(fcm2))
colnames(fcm2)[5] <- rownames(fcm2)[5] <- "fox"
fcm2
fcm_compress(fcm2)

---

dfm_group

Combine documents in a dfm by a grouping variable

Description

Combine documents in a dfm by a grouping variable, which can also be one of the docvars attached to the dfm. This is identical in functionality to using the "groups" argument in dfm.

Usage

dfm_group(x, groups = NULL, fill = FALSE, force = FALSE)

Arguments

- **x**: a dfm
- **groups**: either: a character vector containing the names of document variables to be used for grouping; or a factor or object that can be coerced into a factor equal in length or rows to the number of documents. See groups for details.
- **fill**: logical; if TRUE and groups is a factor, then use all levels of the factor when forming the new "documents" of the grouped dfm. This will result in documents with zero feature counts for levels not observed. Has no effect if the groups variable(s) are not factors.
- **force**: logical; if TRUE, group by summing existing counts, even if the dfm has been weighted. This can result in invalid sums, such as adding log counts (when a dfm has been weighted by "logcount" for instance using dfm_weight). Does not apply to the term weight schemes "count" and "prop".

Value

dfm_group returns a dfm whose documents are equal to the unique group combinations, and whose cell values are the sums of the previous values summed by group. Document-level variables that have no variation within groups are saved in docvars. Document-level variables that are lists are dropped from grouping, even when these exhibit no variation within groups.

Setting the fill = TRUE offers a way to "pad" a dfm with document groups that may not have been observed, but for which an empty document is needed, for various reasons. If groups is a factor of dates, for instance, then using fill = TRUE ensures that the new documents will consist of one row of the dfm per date, regardless of whether any documents previously existed with that date.
**Examples**

corp <- corpus(c("a a b", "a b c c", "a c d d", "a c c d"),
docvars = data.frame(grp = c("grp1", "grp1", "grp2", "grp2")))
dfmat <- dfm(corp)
dfm_group(dfmat, groups = "grp")
dfm_group(dfmat, groups = c(1, 1, 2, 2))

# equivalent
dfm(dfmat, groups = "grp")
dfm(dfmat, groups = c(1, 1, 2, 2))

---

**dfm_lookup**  
*Apply a dictionary to a dfm*

**Description**

Apply a dictionary to a dfm by looking up all dfm features for matches in a set of dictionary values, and replace those features with a count of the dictionary’s keys. If exclusive = FALSE then the behaviour is to apply a "thesaurus", where each value match is replaced by the dictionary key, converted to capitals if capkeys = TRUE (so that the replacements are easily distinguished from features that were terms found originally in the document).

**Usage**

dfm_lookup(  
  x,  
  dictionary,  
  levels = 1:5,  
  exclusive = TRUE,  
  valuetype = c("glob", "regex", "fixed"),  
  case_insensitive = TRUE,  
  capkeys = !exclusive,  
  nomatch = NULL,  
  verbose = quanteda_options("verbose")
)

**Arguments**

- `x` the dfm to which the dictionary will be applied
- `dictionary` a dictionary class object
- `levels` levels of entries in a hierarchical dictionary that will be applied
- `exclusive` if TRUE, remove all features not in dictionary, otherwise, replace values in dictionary with keys while leaving other features unaffected
- `valuetype` the type of pattern matching: "glob" for "glob"-style wildcard expressions; "regex" for regular expressions; or "fixed" for exact matching. See `valuetype` for details.
dfm_lookup

case_insensitive
ignore the case of dictionary values if TRUE

capkeys
if TRUE, convert dictionary keys to uppercase to distinguish them from other features

nomatch
an optional character naming a new feature that will contain the counts of features of x not matched to a dictionary key. If NULL (default), do not tabulate unmatched features.

verbose
print status messages if TRUE

Note
If using dfm_lookup with dictionaries containing multi-word values, matches will only occur if the features themselves are multi-word or formed from ngrams. A better way to match dictionary values that include multi-word patterns is to apply tokens_lookup to the tokens, and then construct the dfm.

See Also
dfm_replace

Examples
dict <- dictionary(list(christmas = c("Christmas", "Santa", "holiday"),
  opposition = c("Opposition", "reject", "notincorpus"),
  taxglob = "tax*",
  taxregex = "tax.+$",
  country = c("United_States", "Sweden")))
dfmat <- dfm(c("My Christmas was ruined by your opposition tax plan.",
  "Does the United_States or Sweden have more progressive taxation?"),
  remove = stopwords("english"))
dfmat

# glob format
dfm_lookup(dfmat, dict, valuetype = "glob")
dfm_lookup(dfmat, dict, valuetype = "glob", case_insensitive = FALSE)

# regex v. glob format: note that "united_states" is a regex match for "tax*"
dfm_lookup(dfmat, dict, valuetype = "glob")
dfm_lookup(dfmat, dict, valuetype = "regex", case_insensitive = TRUE)

# fixed format: no pattern matching
dfm_lookup(dfmat, dict, valuetype = "fixed")
dfm_lookup(dfmat, dict, valuetype = "fixed", case_insensitive = FALSE)

# show unmatched tokens
dfm_lookup(dfmat, dict, nomatch = "_UNMATCHED")
dfm_match

Match the feature set of a dfm to given feature names

Description

Match the feature set of a dfm to a specified vector of feature names. For existing features in x for which there is an exact match for an element of features, these will be included. Any features in x not features will be discarded, and any feature names specified in features but not found in x will be added with all zero counts.

Usage

dfm_match(x, features)

Arguments

x

a dfm

features

character; the feature names to be matched in the output dfm

Details

Selecting on another dfm’s featnames is useful when you have trained a model on one dfm, and need to project this onto a test set whose features must be identical. It is also used in bootstrap_dfm.

Value

A dfm whose features are identical to those specified in features.

Note

Unlike dfm_select, this function will add feature names not already present in x. It also provides only fixed, case-sensitive matches. For more flexible feature selection, see dfm_select.

See Also

dfm_select

Examples

# matching a dfm to a feature vector
dfm_match(dfm(""), letters[1:5])
dfm_match(data_dfm_lbgexample, c("A", "B", "Z"))
dfm_match(data_dfm_lbgexample, c("B", "newfeat1", "A", "newfeat2"))

# matching one dfm to another
txt <- c("This is text one", "The second text", "This is text three")
(dfmat1 <- dfm(txt[1:2]))
(dfmat2 <- dfm(txt[2:3]))
dfm_replace <- dfm_match(dfmat1, featnames(dfmat2))
setequal(featnames(dfmat2), featnames(dfmat3))

---

**dfm_replace**

Replace features in dfm

---

**Description**

Substitute features based on vectorized one-to-one matching for lemmatization or user-defined stemming.

**Usage**

```r
dfm_replace(
  x,
  pattern,
  replacement,
  case_insensitive = TRUE,
  verbose = quanteda_options("verbose")
)
```

**Arguments**

- `x` `dfm` whose features will be replaced
- `pattern` a character vector. See `pattern` for more details.
- `replacement` if `pattern` is a character vector, then `replacement` must be character vector of equal length, for a 1:1 match.
- `case_insensitive` ignore case when matching, if `TRUE`
- `verbose` print status messages if `TRUE`

**Examples**

```r
dfmat1 <- dfm(data_corpus_irishbudget2010)

# lemmatization
lis <- c("foci", "focus", "focused", "focuses", "focusing", "focussed", "focusses")
lemma <- rep("focus", length(lis))

dfmat2 <- dfm_replace(dfmat1, pattern = lis, replacement = lemma)
featnames(dfm_select(dfmat2, pattern = lis))

# stemming
feat <- featnames(dfmat1)
featstem <- char_wordstem(feat, "porter")
dfmat3 <- dfm_replace(dfmat1, pattern = feat, replacement = featstem, case_insensitive = FALSE)
identical(dfmat3, dfm_wordstem(dfmat1, "porter"))
```
dfm_sample

Randomly sample documents or features from a dfm

Description
Sample randomly from a dfm object, from documents or features.

Usage
dfm_sample(
  x,
  size = ifelse(margin == "documents", ndoc(x), nfeat(x)),
  replace = FALSE,
  prob = NULL,
  margin = c("documents", "features")
)

Arguments

  x  the dfm object whose documents or features will be sampled

  size  a positive number, the number of documents or features to select. The default is
        the number of documents or the number of features, for margin = "documents"
        and margin = "features" respectively.

  replace  logical; should sampling be with replacement?

  prob  a vector of probability weights for obtaining the elements of the vector being
        sampled.

  margin  dimension (of a dfm) to sample: can be documents or features

Value
A dfm object with number of documents or features equal to size, drawn from the dfm x.

See Also

  sample

Examples
set.seed(10)
dfmat <- dfm(c("a b c c d", "a a c c d d d"))
head(dfmat)
head(dfm_sample(dfmat))
head(dfm_sample(dfmat, replace = TRUE))
head(dfm_sample(dfmat, margin = "features"))
head(dfm_sample(dfmat, margin = "features", replace = TRUE))
**dfm_select**

Select features from a dfm or fcm

**Description**

This function selects or removes features from a dfm or fcm, based on feature name matches with pattern. The most common usages are to eliminate features from a dfm already constructed, such as stopwords, or to select only terms of interest from a dictionary.

**Usage**

```r
dfm_select(
  x, 
  pattern = NULL, 
  selection = c("keep", "remove"), 
  valuetype = c("glob", "regex", "fixed"), 
  case_insensitive = TRUE, 
  min_nchar = NULL, 
  max_nchar = NULL, 
  verbose = quanteda_options("verbose")
)

dfm_remove(x, ...)

dfm_keep(x, ...)

fcm_select(
  x, 
  pattern = NULL, 
  selection = c("keep", "remove"), 
  valuetype = c("glob", "regex", "fixed"), 
  case_insensitive = TRUE, 
  verbose = quanteda_options("verbose"),
  ...
)

fcm_remove(x, pattern = NULL, ...)

fcm_keep(x, pattern = NULL, ...)
```

**Arguments**

- **x**  
  the dfm or fcm object whose features will be selected
- **pattern**  
  a character vector, list of character vectors, dictionary, or collocations object. See pattern for details.
- **selection**  
  whether to keep or remove the features
valuetype the type of pattern matching: "glob" for "glob"-style wildcard expressions; "regex" for regular expressions; or "fixed" for exact matching. See valuetype for details.

For dfm_select, pattern may also be a dfm; see Value below.

case_insensitive ignore the case of dictionary values if TRUE

min_nchar, max_nchar optional numerics specifying the minimum and maximum length in characters for tokens to be removed or kept; defaults are NULL for no limits. These are applied after (and hence, in addition to) any selection based on pattern matches.

verbose if TRUE print message about how many pattern were removed

... used only for passing arguments from dfm_remove or dfm_keep to dfm_select. Cannot include selection.

Details
dfm_remove and fcm_remove are simply a convenience wrappers to calling dfm_select and fcm_select with selection = "remove".
dfm_keep and fcm_keep are simply a convenience wrappers to calling dfm_select and fcm_select with selection = "keep".

Value
A dfm or fcm object, after the feature selection has been applied.

For compatibility with earlier versions, when pattern is a dfm object and selection = "keep", then this will be equivalent to calling dfm_match. In this case, the following settings are always used: case_insensitive = FALSE, and valuetype = "fixed". This functionality is deprecated, however, and you should use dfm_match instead.

Note
This function selects features based on their labels. To select features based on the values of the document-feature matrix, use dfm_trim.

See Also
dfm_match

Examples
dfmat <- dfm(c("My Christmas was ruined by your opposition tax plan.",
    "Does the United_States or Sweden have more progressive taxation?"),
    tolower = FALSE)
dict <- dictionary(list(countries = c("United_States", "Sweden", "France"),
    wordsEndingInY = c("by", "my"),
    notintext = "blahblah"))
dfm_select(dfmat, pattern = dict)
dfm_select(dfmat, pattern = dict, case_insensitive = FALSE)
dfm_sort

Sort a dfm by frequency of one or more margins

Description

Sorts a dfm by descending frequency of total features, total features in documents, or both.

Usage

dfm_sort(x, decreasing = TRUE, margin = c("features", "documents", "both"))

Arguments

x Document-feature matrix created by dfm
decreasing logical; if TRUE, the sort will be in descending order, otherwise sort in increasing order
margin which margin to sort on: features to sort by frequency of features, documents to sort by total feature counts in documents, and both to sort by both

Value

A sorted dfm matrix object

Author(s)

Ken Benoit
Examples

defmat <- dfm(data_corpus_inaugural)
head(dfmat)
head(dfm_sort(dfmat))
head(dfm_sort(dfmat, decreasing = FALSE, "both"))

---

dfm_subset

Extract a subset of a dfm

Description

Returns document subsets of a dfm that meet certain conditions, including direct logical operations on docvars (document-level variables). dfm_subset functions identically to `subset.data.frame`, using non-standard evaluation to evaluate conditions based on the docvars in the dfm.

Usage

dfm_subset(x, subset, select, ...)

Arguments

- `x`  
  dfm object to be subsetted
- `subset`  
  logical expression indicating the documents to keep: missing values are taken as false
- `select`  
  expression, indicating the docvars to select from the dfm; or a dfm object, in which case the returned dfm will contain the same documents as the original dfm, even if these are empty. See Details.
- `...`  
  not used

Details

To select or subset features, see `dfm_select` instead.

When `select` is a dfm, then the returned dfm will be equal in document dimension and order to the dfm used for selection. This is the document-level version of using `dfm_select` where pattern is a dfm: that function matches features, while dfm_subset will match documents.

Value

dfm object, with a subset of documents (and docvars) selected according to arguments

See Also

`subset.data.frame`
Examples

corp <- corpus(c(d1 = "a b c d", d2 = "a a b e",
                 d3 = "b b c e", d4 = "e e f a b"),
               docvars = data.frame(grp = c(1, 1, 2, 3)))
dfmat <- dfm(corp)
# selecting on a docvars condition
dfm_subset(dfmat, grp > 1)
# selecting on a supplied vector
dfm_subset(dfmat, c(TRUE, FALSE, TRUE, FALSE))

# selecting on a dfm
dfm1 <- dfm(c(d1 = "a b b c", d2 = "b b c d"))
dfm2 <- dfm(c(d1 = "x y z", d2 = "a b c c d", d3 = "x x x"))
dfm_subset(dfm1, subset = dfm2)
dfm_subset(dfm1, subset = dfm2[c(3,1,2), ])}

---

dfm_tfidf Weight a dfm by tf-idf

Description

Weight a dfm by term frequency-inverse document frequency (tf-idf), with full control over options. Uses fully sparse methods for efficiency.

Usage

dfm_tfidf(
  x,
  scheme_tf = "count",
  scheme_df = "inverse",
  base = 10,
  force = FALSE,
  ...
)

Arguments

x object for which idf or tf-idf will be computed (a document-feature matrix)
scheme_tf scheme for dfm_weight; defaults to "count"
scheme_df scheme for docfreq; defaults to "inverse". Other options to docfreq can be passed through the ellipsis (...).
base the base for the logarithms in the tf and docfreq calls; default is 10
force logical; if TRUE, apply weighting scheme even if the dfm has been weighted before. This can result in invalid weights, such as as weighting by "prop" after applying "logcount", or after having grouped a dfm using dfm_group.
... additional arguments passed to docfreq.
dfm_tfidf computes term frequency-inverse document frequency weighting. The default is to use counts instead of normalized term frequency (the relative term frequency within document), but this can be overridden using `scheme_tf = "prop"`.

**References**


**See Also**

dfm_weight, docfreq

**Examples**

dmat1 <- as.dfm(data_dfm_lbgexample)
head(dmat1[, 5:10])
head(dfm_tfidf(dmat1)[, 5:10])
docfreq(dmat1)[5:15]
head(dfm_weight(dmat1)[, 5:10])

# replication of worked example from
dmat2 <-
  matrix(c(1,1,2,1,0,0, 1,1,0,0,2,3),
         byrow = TRUE, nrow = 2,
         dimnames = list(docs = c("document1", "document2"),
                         features = c("this", "is", "a", "sample",
                                      "another", "example"))) %>%
  as.dfm()
dmat2
docfreq(dmat2)
dfm_tfidf(dmat2, scheme_tf = "prop") %>% round(digits = 2)

## Not run:
# comparison with tm
if (requireNamespace("tm")) {
  convert(dmat2, to = "tm") %>% tm::weightTfIdf() %>% as.matrix()
  # same as:
  dfm_tfidf(dmat2, base = 2, scheme_tf = "prop")
}

## End(Not run)
**dfm_tolower**  

*Convert the case of the features of a dfm and combine*

**Description**

*dfm_tolower* and *dfm_toupper* convert the features of the dfm or fcm to lower and upper case, respectively, and then recombine the counts.

**Usage**

- *dfm_tolower(x, keep_acronyms = FALSE)*
- *dfm_toupper(x)*
- *fcm_tolower(x, keep_acronyms = FALSE)*
- *fcm_toupper(x)*

**Arguments**

- **x**  
  the input object whose character/tokens/feature elements will be case-converted
- **keep_acronyms**  
  logical; if TRUE, do not lowercase any all-uppercase words (applies only to *_tolower functions*)

**Details**

*fcm_tolower* and *fcm_toupper* convert both dimensions of the fcm to lower and upper case, respectively, and then recombine the counts. This works only on fcm objects created with context = "document".

**Examples**

- # for a document-feature matrix  
  dfmat <- dfm(c("b A A", "C C a b B"), tolower = FALSE)  
  dfmat  
  dfm_tolower(dfmat)  
  dfm_toupper(dfmat)

- # for a feature co-occurrence matrix  
  fcmat <- fcm(tokens(c("b A A d", "C C a b B e"),  
  context = "document")  
  fcmat  
  fcm_tolower(fcmat)  
  fcm_toupper(fcmat)
dfm_trim

Trim a dfm using frequency threshold-based feature selection

Description

Returns a document by feature matrix reduced in size based on document and term frequency, usually in terms of a minimum frequency, but may also be in terms of maximum frequencies. Setting a combination of minimum and maximum frequencies will select features based on a range. Feature selection is implemented by considering features across all documents, by summing them for term frequency, or counting the documents in which they occur for document frequency. Rank and quantile versions of these are also implemented, for taking the first \( n \) features in terms of descending order of overall global counts or document frequencies, or as a quantile of all frequencies.

Usage

dfm_trim(
  x,
  min_termfreq = NULL,
  max_termfreq = NULL,
  termfreq_type = c("count", "prop", "rank", "quantile"),
  min_docfreq = NULL,
  max_docfreq = NULL,
  docfreq_type = c("count", "prop", "rank", "quantile"),
  sparsity = NULL,
  verbose = quanteda_options("verbose"),
  ...
)

Arguments

- `x` a dfm object
- `min_termfreq`, `max_termfreq` minimum/maximum values of feature frequencies across all documents, below/above which features will be removed
- `termfreq_type` how `min_termfreq` and `max_termfreq` are interpreted. "count" sums the frequencies; "prop" divides the term frequencies by the total sum; "rank" is matched against the inverted ranking of features in terms of overall frequency, so that 1, 2, ... are the highest and second highest frequency features, and so on; "quantile" sets the cutoffs according to the quantiles (see `quantile`) of term frequencies.
- `min_docfreq`, `max_docfreq` minimum/maximum values of a feature’s document frequency, below/above which features will be removed
- `docfreq_type` specify how `min_docfreq` and `max_docfreq` are interpreted. "count" is the same as `docfreq(x, scheme = "count")`; "prop" divides the document frequencies by the total sum; "rank" is matched against the inverted ranking of
document frequency, so that 1, 2, ... are the features with the highest and second highest document frequencies, and so on; “quantile” sets the cutoffs according to the quantiles (see quantile) of document frequencies.

sparsity equivalent to 1 - min_docfreq, included for comparison with tm
verbose print messages
... not used

Value

A dfm reduced in features (with the same number of documents)

Note

Trimming a dfm object is an operation based on the values in the document-feature matrix. To select subsets of a dfm based on the features themselves (meaning the feature labels from featnames) – such as those matching a regular expression, or removing features matching a stopword list, use dfm_select.

See Also

dfm_select, dfm_sample

Examples

(dfmat <- dfm(data_corpus_inaugural[1:5]))

# keep only words occurring >= 10 times and in >= 2 documents
dfm_trim(dfmat, min_termfreq = 10, min_docfreq = 2)

# keep only words occurring >= 10 times and in at least 0.4 of the documents
dfm_trim(dfmat, min_termfreq = 10, min_docfreq = 0.4)

# keep only words occurring <= 10 times and in <=2 documents
dfm_trim(dfmat, max_termfreq = 10, max_docfreq = 2)

# keep only words occurring <= 10 times and in at most 3/4 of the documents
dfm_trim(dfmat, max_termfreq = 10, max_docfreq = 0.75)

# keep only words occurring 5 times in 1000, and in 2 of 5 of documents
dfm_trim(dfmat, min_docfreq = 0.4, min_termfreq = 0.005, termfreq_type = "prop")

# keep only words occurring frequently (top 20%) and in <=2 documents
dfm_trim(dfmat, min_termfreq = 0.2, max_docfreq = 2, termfreq_type = "quantile")

## Not run:
# compare to removeSparseTerms from the tm package
(dfmattm <- convert(dfmat, "tm"))
tm::removeSparseTerms(dfmattm, 0.7)
dfm_trim(dfmat, min_docfreq = 0.3)
dfm_trim(dfmat, sparsity = 0.7)
dfm_weight

Weight the feature frequencies in a dfm

### Usage

```r
dfm_weight(
  x,
  scheme = c("count", "prop", "propmax", "logcount", "boolean", "augmented", "logave"),
  weights = NULL,
  base = 10,
  K = 0.5,
  force = FALSE
)
```

dfm_smooth(x, smoothing = 1)

### Arguments

- `x`: document-feature matrix created by `dfm`
- `scheme`: a label of the weight type:
  - `count`: `tf_{ij}`, an integer feature count (default when a dfm is created)
  - `prop`: the proportion of the feature counts of total feature counts (aka relative frequency), calculated as `tf_{ij} / \sum_j tf_{ij}`
  - `propmax`: the proportion of the feature counts of the highest feature count in a document, `tf_{ij} / \max_j tf_{ij}`
  - `logcount`: take the `1 + the logarithm of each count, for the given base, or 0 if the count was zero: `1 + \log_{base}(tf_{ij})` if `tf_{ij} > 0`, or 0 otherwise.
  - `boolean`: recode all non-zero counts as 1
  - `augmented`: equivalent to `K + (1 - K) * dfm_weight(x,"propmax")`
  - `logave`: `1 + the log of the counts) / (1 + log of the counts / the average count within document), or` 
    
    \[
    \frac{1 + \log_{base}(tf_{ij})}{1 + \log_{base}(\sum_j tf_{ij}/N_i)}
    \]
- `weights`: if `scheme` is unused, then weights can be a named numeric vector of weights to be applied to the dfm, where the names of the vector correspond to feature labels of the dfm, and the weights will be applied as multipliers to the existing feature counts for the corresponding named features. Any features not named will be assigned a weight of 1.0 (meaning they will be unchanged).
**dfm_weight**

- **base**  
  base for the logarithm when scheme is "logcount" or logave  
- **K**  
  the K for the augmentation when scheme = "augmented"  
- **force**  
  logical; if TRUE, apply weighting scheme even if the dfm has been weighted before. This can result in invalid weights, such as weighting by "prop" after applying "logcount", or after having grouped a dfm using dfm_group.  
- **smoothing**  
  constant added to the dfm cells for smoothing, default is 1

**Value**

dfm_weight returns the dfm with weighted values. Note the because the default weighting scheme is "count", simply calling this function on an unweighted dfm will return the same object. Many users will want the normalized dfm consisting of the proportions of the feature counts within each document, which requires setting scheme = "prop".  
dfm_smooth returns a dfm whose values have been smoothed by adding the smoothing amount. Note that this effectively converts a matrix from sparse to dense format, so may exceed memory requirements depending on the size of your input matrix.

**References**


**See Also**

dfm_tfidf, docfreq

**Examples**

dfm1 <- dfm(data_corpus_inaugural)
dfm2 <- dfm_weight(dfm1, scheme = "prop")
topfeatures(dfm2)
dfm3 <- dfm_weight(dfm1)
topfeatures(dfm3)
dfm4 <- dfm_weight(dfm1, scheme = "logcount")
topfeatures(dfm4)
dfm5 <- dfm_weight(dfm1, scheme = "logave")
topfeatures(dfm5)

# combine these methods for more complex dfm_weightings, e.g. as in Section 6.4 of Introduction to Information Retrieval
head(dfm_tfidf(dfm1, scheme_tf = "logcount"))

# apply numeric weights
str <- c("apple is better than banana", "banana banana apple much better")
(dfm6 <- dfm(str, remove = stopwords("english")))
dfm_weight(dfm6, weights = c(apple = 5, banana = 3, much = 0.5))

# smooth the dfm
dictionary <- dfm(data_corpus_inaugural)
dfm_smooth(dictionary, 0.5)

---

dictionary

Create a dictionary

Description

Create a **quanteda** dictionary class object, either from a list or by importing from a foreign format. Currently supported input file formats are the WordStat, LIWC, Lexicoder v2 and v3, and Yoshikoder formats. The import using the LIWC format works with all currently available dictionary files supplied as part of the LIWC 2001, 2007, and 2015 software (see References).

Usage

dictionary(
  x,
  file = NULL,
  format = NULL,
  separator = " ",
  tolower = TRUE,
  encoding = "auto"
)

Arguments

- **x**: a named list of character vector dictionary entries, including `valuetype` pattern matches, and including multi-word expressions separated by `concatenator`. See examples. This argument may be omitted if the dictionary is read from file.
- **file**: file identifier for a foreign dictionary
- **format**: character identifier for the format of the foreign dictionary. If not supplied, the format is guessed from the dictionary file’s extension. Available options are:
  - "wordstat" format used by Provalis Research’s WordStat software
  - "LIWC" format used by the Linguistic Inquiry and Word Count software
  - "yoshikoder" format used by Yoshikoder software
  - "lexicoder" format used by Lexicoder
  - "YAML" the standard YAML format
- **separator**: the character in between multi-word dictionary values. This defaults to " ".
- **tolower**: if TRUE, convert all dictionary values to lowercase
- **encoding**: additional optional encoding value for reading in imported dictionaries. This uses the `iconv` labels for encoding. See the “Encoding” section of the help for file.
Details

Dictionaries can be subsetted using \[ \] and \[[\], operating the same as the equivalent \list \] operators.
Dictionaries can be coerced from lists using \as \.dictionary\, coerced to named lists of characters using \as \.list\, and checked using \is \.dictionary\.

Value

A dictionary class object, essentially a specially classed named list of characters.

References

Yoshikoder page, from Will Lowe http://conjugateprior.org/software/yoshikoder/.
Lexicoder format, http://www.lexicoder.com

See Also

dfm, \as \.dictionary, \as \.list, \is \.dictionary

Examples

corp <- corpus_subset(data_corpus_inaugural, Year>1900)
dict <- dictionary(list(christmas = c("Christmas", "Santa", "holiday"),
opposition = c("Opposition", "reject", "notincorpus"),
taxing = "taxing",
taxation = "taxation",
taxregex = "tax*",
country = "america"))
head(dfm(corp, dictionary = dict))

# subset a dictionary
dict[1:2]
dict[c("christmas", "opposition")]
dict[["opposition"]]

# combine dictionaries
c(dict["christmas"], dict["country"])

## Not run:
# import the Laver-Garry dictionary from Provalis Research
dictfile <- tempfile()
download.file("https://provalisresearch.com/Download/LaverGarry.zip",
dictfile, mode = "wb")
unzip(dictfile, exdir = (td <- tempdir()))
dictlg <- dictionary(file = paste(td, "LaverGarry.cat", sep = "/"))
head(dfm(data_corpus_inaugural, dictionary = dictlg))
# import a LIWC formatted dictionary from http://www.moralfoundations.org
download.file("https://goo.gl/5gmwXq", tf <- tempfile())
dictliwc <- dictionary(file = tf, format = "LIWC")
head(dfm(data_corpus_inaugural, dictionary = dictliwc))

## End(Not run)

### docfreq

**Compute the (weighted) document frequency of a feature**

**Description**

For a dfm object, returns a (weighted) document frequency for each term. The default is a simple count of the number of documents in which a feature occurs more than a given frequency threshold. (The default threshold is zero, meaning that any feature occurring at least once in a document will be counted.)

**Usage**

```
docfreq(
  x, 
  scheme = c("count", "inverse", "inversemax", "inverseprob", "unary"), 
  smoothing = 0, 
  k = 0, 
  base = 10, 
  threshold = 0, 
  use.names = TRUE
)
```

**Arguments**

- **x**: a dfm
- **scheme**: type of document frequency weighting, computed as follows, where \( N \) is defined as the number of documents in the dfm and \( s \) is the smoothing constant:
  - count: \( df_j \), the number of documents for which \( n_{ij} > \text{threshold} \)
  - inverse: \( \log_{\text{base}} \left( s + \frac{N}{k + df_j} \right) \)
  - inversemax: \( \log_{\text{base}} \left( s + \max(df_j) \frac{k + df_j}{k + df_j} \right) \)
  - inverseprob: \( \log_{\text{base}} \left( \frac{N - df_j}{k + df_j} \right) \)
docfreq

unary 1 for each feature

smoothing

added to the quotient before taking the logarithm

k

added to the denominator in the "inverse" weighting types, to prevent a zero
document count for a term

base

the base with respect to which logarithms in the inverse document frequency
weightings are computed; default is 10 (see Manning, Raghavan, and Schütze
2008, p123).

threshold

numeric value of the threshold above which a feature will considered in the
computation of document frequency. The default is 0, meaning that a feature’s
document frequency will be the number of documents in which it occurs greater
than zero times.

use.names

logical; if TRUE attach feature labels as names of the resulting numeric vector

... not used

Value

a numeric vector of document frequencies for each feature

References

pdf

Examples

dfmat1 <- dfm(data_corpus_inaugural[1:2])
docfreq(dfmat1[, 1:20])

# replication of worked example from
dfmat2 <-
  matrix(c(1,1,2,1,0,0, 1,1,0,0,2,3),
         byrow = TRUE, nrow = 2,
         dimnames = list(docs = c("document1", "document2"),
                         features = c("this", "is", "a", "sample",
                                     "another", "example"))), %>%
  as.dfm()
dfmat2
docfreq(dfmat2)
docfreq(dfmat2, scheme = "inverse")
docfreq(dfmat2, scheme = "inverse", k = 1, smoothing = 1)
docfreq(dfmat2, scheme = "unary")
docfreq(dfmat2, scheme = "inversemx")
docfreq(dfmat2, scheme = "inverseprob")
docnames

Get or set document names

Description

Get or set the document names of a corpus, tokens, or dfm object.

Usage

\[
docnames(x) \\
docnames(x) \leftarrow value
\]

Arguments

- **x**: the object with docnames
- **value**: a character vector of the same length as \(x\)

Value

- `docnames` returns a character vector of the document names
- `docnames <-` assigns new values to the document names of an object. `docnames` can only be character, so any non-character value assigned to be a docname will be coerced to mode 'character'.

See Also

- `featnames`

Examples

# get and set document names to a corpus
corp <- data_corpus_inaugural
docnames(corp) <- char_tolower(docnames(corp))

# get and set document names to a tokens
toks <- tokens(data_corpus_inaugural)
docnames(toks) <- char_tolower(docnames(toks))

# get and set document names to a dfm
dfmat <- dfm(data_corpus_inaugural[1:5])
docnames(dfmat) <- char_tolower(docnames(dfmat))

# reassign the document names of the inaugural speech corpus
docnames(data_corpus_inaugural) <- paste("Speech", 1:ndoc(data_corpus_inaugural), sep="")
Get or set document-level variables

Description
Get or set variables associated with a document in a corpus, tokens or dfm object.

Usage

```r
docvars(x, field = NULL)
docvars(x, field = NULL) <- value
```

Arguments

- `x`: corpus, tokens, or dfm object whose document-level variables will be read or set
- `field`: string containing the document-level variable name
- `value`: the new values of the document-level variable

Value

docvars returns a data.frame of the document-level variables, dropping the second dimension to form a vector if a single docvar is returned.
docvars<- assigns value to the named field

Note

Reassigning document variables for a tokens or dfm object is allowed, but discouraged. A better, more reproducible workflow is to create your docvars as desired in the corpus, and let these continue to be attached "downstream" after tokenization and forming a document-feature matrix. Recognizing that in some cases, you may need to modify or add document variables to downstream objects, the assignment operator is defined for tokens or dfm objects as well. Use with caution.

Examples

```r
# retrieving docvars from a corpus
head(docvars(data_corpus_inaugural))
tail(docvars(data_corpus_inaugural, "President"), 10)

# assigning document variables to a corpus
corp <- data_corpus_inaugural
docvars(corp, "President") <- paste("prez", 1:ndoc(corp), sep = "")
head(docvars(corp))
```
Create a feature co-occurrence matrix

Description

Create a sparse feature co-occurrence matrix, measuring co-occurrences of features within a user-defined context. The context can be defined as a document or a window within a collection of documents, with an optional vector of weights applied to the co-occurrence counts.

Usage

fcm(
x,  
context = c("document", "window"),  
count = c("frequency", "boolean", "weighted"),  
window = 5L,  
weights = NULL,  
ordered = FALSE,  
tri = TRUE,  
...
)

Arguments

x character, corpus, tokens, or dfm object from which to generate the feature co-occurrence matrix
context the context in which to consider term co-occurrence: "document" for co-occurrence counts within document; "window" for co-occurrence within a defined window of words, which requires a positive integer value for window. Note: if x is a dfm object, then context can only be "document".
count how to count co-occurrences:
   "frequency" count the number of co-occurrences within the context
   "boolean" count only the co-occurrence or not within the context, irrespective of how many times it occurs.
   "weighted" count a weighted function of counts, typically as a function of distance from the target feature. Only makes sense for context = "window".
window positive integer value for the size of a window on either side of the target feature, default is 5, meaning 5 words before and after the target feature
weights a vector of weights applied to each distance from 1:window, strictly decreasing by default; can be a custom-defined vector of the same length as window
ordered if TRUE the number of times that a term appears before or after the target feature are counted separately. Only makes sense for context = "window".
tri if TRUE return only upper triangle (including diagonal). Ignored if ordered = TRUE
... not used here
Details

The function `fcm` provides a very general implementation of a "context-feature" matrix, consisting of a count of feature co-occurrence within a defined context. This context, following Momtazi et. al. (2010), can be defined as the `document`, `sentences` within documents, `syntactic relationships` between features (nouns within a sentence, for instance), or according to a `window`. When the context is a window, a weighting function is typically applied that is a function of distance from the target word (see Jurafsky and Martin 2015, Ch. 16) and ordered co-occurrence of the two features is considered (see Church & Hanks 1990).

`fcm` provides all of this functionality, returning a $V \times V$ matrix (where $V$ is the vocabulary size, returned by `nfeat`). The `tri = TRUE` option will only return the upper part of the matrix.

Unlike some implementations of co-occurrences, `fcm` counts feature co-occurrences with themselves, meaning that the diagonal will not be zero.

`fcm` also provides "boolean" counting within the context of "window", which differs from the counting within "document".

`is.fcm(x)` returns `TRUE` if and only if its `x` is an object of type `fcm`.

Author(s)

Kenneth Benoit (R), Haiyan Wang (R, C++), Kohei Watanabe (C++)

References


Examples

```r
# see http://bit.ly/29b2zoA
txt1 <- "A D A C E A D F E B A C E D"
fcm(txt1, context = "window", window = 2)
fcm(txt1, context = "window", count = "weighted", window = 3)
fcm(txt1, context = "window", count = "weighted", window = 3, weights = c(3, 2, 1), ordered = TRUE, tri = FALSE)

# with multiple documents
txt2 <- c("a a a b b c", "a a c e", "a c e f g")
fcm(txt2, context = "document", count = "frequency")
fcm(txt2, context = "document", count = "boolean")
fcm(txt2, context = "window", window = 2)
```
# from tokens
txt3 <- c("The quick brown fox jumped over the lazy dog.",
          "The dog jumped and ate the fox.")
toks <- tokens(char_tolower(txt3), remove_punct = TRUE)
fcm(toks, context = "document")
fcm(toks, context = "window", window = 3)

---

fcm_sort  
Sort an fcm in alphabetical order of the features

**Description**

Sorts an fcm in alphabetical order of the features.

**Usage**

fcm_sort(x)

**Arguments**

x  
fcm object

**Value**

A fcm object whose features have been alphabetically sorted. Differs from `fcm_sort` in that this function sorts the fcm by the feature labels, not the counts of the features.

**Author(s)**

Kenneth Benoit

**Examples**

# with tri = FALSE
fcmat1 <- fcm(tokens(c("A X Y C B A", "X Y C A B B")), tri = FALSE)
rownames(fcmat1)[3] <- colnames(fcmat1)[3] <- "Z"
fcmat1
fcm_sort(fcmat1)

# with tri = TRUE
fcmat2 <- fcm(tokens(c("A X Y C B A", "X Y C A B B")), tri = TRUE)
rownames(fcmat2)[3] <- colnames(fcmat2)[3] <- "Z"
fcmat2
fcm_sort(fcmat2)
featfreq

Compute the frequencies of features

Description

For a dfm object, returns a frequency for each feature, computed across all documents in the dfm. This is equivalent to colSums(x).

Usage

featfreq(x)

Arguments

x a dfm

Value

a (named) numeric vector of feature frequencies

Examples

dfmat <- dfm(data_char_sampletext)
featfreq(dfmat)

featnames

Get the feature labels from a dfm

Description

Get the features from a document-feature matrix, which are stored as the column names of the dfm object.

Usage

featnames(x)

Arguments

x the dfm whose features will be extracted

Value

character vector of the feature labels
Examples

dfmat <- dfm(data_corpus_inaugural)

# first 50 features (in original text order)
head(featnames(dfmat), 50)

# first 50 features alphabetically
head(sort(featnames(dfmat)), 50)

# contrast with descending total frequency order from topfeatures()
names(topfeatures(dfmat, 50))

---

head.corpus

Return the first or last part of a corpus

Description

For a corpus object, returns the first or last n documents.

Usage

```r
## S3 method for class 'corpus'
head(x, n = 6L, ...)
```

```r
## S3 method for class 'corpus'
tail(x, n = 6L, ...)
```

Arguments

- **x**: a dfm object
- **n**: a single integer. If positive, the number of documents for the resulting object: number of first/last documents for the dfm. If negative, all but the n last/first number of documents of x.
- **...**: additional arguments passed to other functions

Value

A corpus class object corresponding to the subset defined by n.

Examples

```r
head(data_corpus_irishbudget2010, 3) %>% summary()
```

```r
tail(data_corpus_irishbudget2010, 3) %>% summary()
```
head.dfm

Return the first or last part of a dfm

Description

For a dfm object, returns the first or last n documents and first nf features.

Usage

## S3 method for class 'dfm'
head(x, n = 6L, nf = nfeat(x), ...)

## S3 method for class 'dfm'
tail(x, n = 6L, nf = nfeat(x), ...)

Arguments

x  
a dfm object

n  
a single, positive integer. If positive, size for the resulting object: number of first/last documents for the dfm. If negative, all but the n last/first number of documents of x.

nf  
the number of features to return, where the resulting object will contain the first ncol features; default is all features

...  
additional arguments passed to other functions

Value

A dfm class object corresponding to the subset defined by n and nf.

Examples

head(data_dfm_lbgexample, 3, nf = 5)
head(data_dfm_lbgexample, -4)
tail(data_dfm_lbgexample)
tail(data_dfm_lbgexample, n = 3, nf = 4)
kwic  Locate keywords-in-context

Description

For a text or a collection of texts (in a quanteda corpus object), return a list of a keyword supplied by the user in its immediate context, identifying the source text and the word index number within the source text. (Not the line number, since the text may or may not be segmented using end-of-line delimiters.)

Usage

kwic(  
  x,  
  pattern,  
  window = 5,  
  valuetype = c("glob", "regex", "fixed"),  
  separator = " ",  
  case_insensitive = TRUE,  
  ...  
)

is.kwic(x)

Arguments

x a character, corpus, or tokens object  
pattern a character vector, list of character vectors, dictionary, or collocations object. See pattern for details.  
window the number of context words to be displayed around the keyword.  
valuetype the type of pattern matching: "glob" for "glob"-style wildcard expressions; "regex" for regular expressions; or "fixed" for exact matching. See valuetype for details.  
separator character to separate words in the output  

case_insensitive match without respect to case if TRUE  
...

Value

A kwic classed data.frame, with the document name (docname), the token index positions (from and to, which will be the same for single-word patterns, or a sequence equal in length to the number of elements for multi-word phrases), the context before (pre), the keyword in its original format (keyword, preserving case and attached punctuation), and the context after (post). The return object has its own print method, plus some special attributes that are hidden in the print view. If you want to turn this into a simple data.frame, simply wrap the result in data.frame.
**Note**

pattern will be a keyword pattern or phrase, possibly multiple patterns, that may include punctuation. If a pattern contains whitespace, it is best to wrap it in `phrase` to make this explicit. However if pattern is a **collocations** or **dictionary** object, then the collocations or multi-word dictionary keys will automatically be considered phrases where each whitespace-separated element matches a token in sequence.

**Examples**

```r
head(kwic(data_corpus_inaugural, pattern = "secure\*", window = 3, valuetype = "glob"))
head(kwic(data_corpus_inaugural, pattern = "secur", window = 3, valuetype = "regex"))
head(kwic(data_corpus_inaugural, pattern = "security", window = 3, valuetype = "fixed"))

toks <- tokens(data_corpus_inaugural)
kwic(data_corpus_inaugural, pattern = phrase("war against"))
kwic(data_corpus_inaugural, pattern = phrase("war against"), valuetype = "regex")

kw <- kwic(data_corpus_inaugural, "provident\*")
is.kwic(kw)
is.kwic("Not a kwic")
is.kwic(kw[, c("pre", "post")])
```

---

**metacorpus**  
*Get or set corpus metadata*

**Description**

Get or set the corpus-level metadata in a **corpus** object.

**Usage**

```r
metacorpus(x, field = NULL)
metacorpus(x, field) <- value
```

**Arguments**

- `x`  
a **corpus** object
- `field`  
metadata field name(s); if NULL (default), return all metadata names
- `value`  
new value of the corpus metadata field

**Value**

- For `metacorpus`, a named list of the metadata fields in the corpus.
- For `metacorpus <-`, the corpus with the updated metadata.
Examples

```r
metacorpus(data_corpus_inaugural)
metacorpus(data_corpus_inaugural, "source")
metacorpus(data_corpus_inaugural, "citation") <- "Presidential Speeches Online Project (2014)."
metacorpus(data_corpus_inaugural, "citation")
```

---

**metadoc**

*Get or set document-level meta-data*

**Description**

Get or set document-level meta-data. Document-level meta-data are a special type of **docvars**, meant to contain information about documents that would not be used as a "variable" for analysis. An example could be the source of the document, or notes pertaining to its transformation, copyright information, etc.

Document-level meta-data differs from corpus-level meta-data in that the latter pertains to the collection of texts as a whole, whereas the document-level version can differ with each document.

**Usage**

```r
metadoc(x, field = NULL)
metadoc(x, field = NULL) <- value
```

**Arguments**

- **x**: a corpus object
- **field**: character, the name of the metadata field(s) to be queried or set
- **value**: the new value of the new meta-data field

**Value**

- For **texts**, a character vector of the texts in the corpus.
- For **texts <-**, the corpus with the updated texts.

**Note**

Document-level meta-data names are preceded by an underscore character, such as `_language`, but when named in in the `field` argument, do *not* need the underscore character.

**See Also**

- **metacorpus**
Examples

```r
mycorp <- corpus_subset(data_corpus_inaugural, Year > 1990)
summary(mycorp, showmeta = TRUE)
metadoc(mycorp, "encoding") <- "UTF-8"
metadoc(mycorp)
metadoc(mycorp, "language") <- "english"
summary(mycorp, showmeta = TRUE)
```

ndoc  

*Count the number of documents or features*

Description

Get the number of documents or features in an object.

Usage

```r
ndoc(x)
nfeat(x)
```

Arguments

x  
a *quanteda* object: a *corpus*, *dfm*, or *tokens* object, or a readtext object from the *readtext* package.

Details

*ndoc* returns the number of documents in an object whose texts are organized as "documents" (a *corpus*, *dfm*, or *tokens* object, a readtext object from the *readtext* package).

*nfeat* returns the number of features from a *dfm*; it is an alias for *ntype* when applied to *dfm* objects. This function is only defined for *dfm* objects because only these have "features". (To count tokens, see *ntoken*.)

Value

an integer (count) of the number of documents or features

See Also

*ntoken*
Examples

# number of documents
ndoc(data_corpus_inaugural)
ndoc(corpus_subset(data_corpus_inaugural, Year > 1980))
ndoc(tokens(data_corpus_inaugural))
ndoc(dfm(corpus_subset(data_corpus_inaugural, Year > 1980)))

# number of features
nfeat(dfm(corpus_subset(data_corpus_inaugural, Year > 1980), remove_punct = FALSE))
nfeat(dfm(corpus_subset(data_corpus_inaugural, Year > 1980), remove_punct = TRUE))

---

nscrabble | Count the Scrabble letter values of text

Description

Tally the Scrabble letter values of text given a user-supplied function, such as the sum (default) or mean of the character values.

Usage

nscrabble(x, FUN = sum)

Arguments

- **x**: a character vector
- **FUN**: function to be applied to the character values in the text; default is sum, but could also be mean or a user-supplied function

Value

a (named) integer vector of Scrabble letter values, computed using FUN, corresponding to the input text(s)

Note

Character values are only defined for non-accented Latin a-z, A-Z letters. Lower-casing is unnecessary.

We would be happy to add more languages to this extremely useful function if you send us the values for your language!

Author(s)

Kenneth Benoit

Examples

nscrabble(c("muzjiks", "excellency"))
nscrabble(data_corpus_inaugural[1:5], mean)
nsentence

Count the number of sentences

Description

Return the count of sentences in a corpus or character object.

Usage

nsentence(x, …)

Arguments

x  a character or corpus whose sentences will be counted
...
additional arguments passed to tokens

Value

count(s) of the total sentences per text

Note

nsentence() relies on the boundaries definitions in the stringi package (see stri_opts_brkiter). It does not count sentences correctly if the text has been transformed to lower case, and for this reason nsentence() will issue a warning if it detects all lower-cased text.

Examples

# simple example
txt <- c(text1 = "This is a sentence: second part of first sentence.",
text2 = "A word. Repeated repeated.",
text3 = "Mr. Jones has a PhD from the LSE. Second sentence.")
nsentence(txt)

nsyllable

Count syllables in a text

Description

Returns a count of the number of syllables in texts. For English words, the syllable count is exact and looked up from the CMU pronunciation dictionary, from the default syllable dictionary data_int_syllables. For any word not in the dictionary, the syllable count is estimated by counting vowel clusters.

data_int_syllables is a quanteda-supplied data object consisting of a named numeric vector of syllable counts for the words used as names. This is the default object used to count English syllables. This object that can be accessed directly, but we strongly encourage you to access it only through the nsyllable() wrapper function.
Usage

```r	nsyllable(
    x,
    syllable_dictionary = quanteda::data_int_syllables,
    use.names = FALSE
)
```

Arguments

- `x` character vector or tokens object whose syllables will be counted. This will count all syllables in a character vector without regard to separating tokens, so it is recommended that `x` be individual terms.
- `syllable_dictionary` optional named integer vector of syllable counts where the names are lower case tokens. When set to `NULL` (default), then the function will use the quanteda data object `data_int_syllables`, an English pronunciation dictionary from CMU.
- `use.names` logical; if `TRUE`, assign the tokens as the names of the syllable count vector

Value

If `x` is a character vector, a named numeric vector of the counts of the syllables in each element. If `x` is a tokens object, return a list of syllable counts where each list element corresponds to the tokens in a document.

Note

All tokens are automatically converted to lowercase to perform the matching with the syllable dictionary, so there is no need to perform this step prior to calling `nsyllable()`.

`nsyllable()` only works reliably for English, as the only syllable count dictionary we could find is the freely available CMU pronunciation dictionary at http://www.speech.cs.cmu.edu/cgi-bin/cmudict. If you have a dictionary for another language, please email the package maintainer as we would love to include it.

Examples

```r
# character	nsyllable(c("cat", "syllable", "supercalifragilisticexpialidocious",
    "Brexit", "Administration"), use.names = TRUE)

# tokens
txt <- c(doc1 = "This is an example sentence.",
    doc2 = "Another of two sample sentences."
nsyllable(tokens(txt, remove_punct = TRUE))
# punctuation is not counted
nsyllable(tokens(txt), use.names = TRUE)
```
ntoken  Count the number of tokens or types

Description
Get the count of tokens (total features) or types (unique tokens).

Usage
ntoken(x, ...)
ntype(x, ...)

Arguments
x a quanteda object: a character, corpus, tokens, or dfm object
... additional arguments passed to tokens

Details
The precise definition of "tokens" for objects not yet tokenized (e.g. character or corpus objects) can be controlled through optional arguments passed to tokens through ....
For dfm objects, ntype will only return the count of features that occur more than zero times in the dfm.

Value
named integer vector of the counts of the total tokens or types

Note
Due to differences between raw text tokens and features that have been defined for a dfm, the counts may be different for dfm objects and the texts from which the dfm was generated. Because the method tokenizes the text in order to count the tokens, your results will depend on the options passed through to tokens.

Examples
# simple example
txt <- c(text1 = "This is a sentence, this.", text2 = "A word. Repeated repeated.")
ntoken(txt)
ntype(txt)
ntoken(char_tolower(txt))  # same
ntype(char_tolower(txt))  # fewer types
ntoken(char_tolower(txt), remove_punct = TRUE)
ntype(char_tolower(txt), remove_punct = TRUE)

# with some real texts
phrase

Declare a compound character to be a sequence of separate pattern
matches

Description

Declares that a whitespace-separated expression consists of multiple patterns, separated by whitespace. This is typically used as a wrapper around pattern to make it explicit that the pattern elements are to be used for matches to multi-word sequences, rather than individual, unordered matches to single words.

Usage

phrase(x)

is.phrase(x)

Arguments

x the sequence, as a character object containing whitespace separating the patterns

Value

phrase returns a specially classed list whose white-spaced elements have been parsed into separate character elements.

is.phrase returns TRUE if the object was created by phrase; FALSE otherwise.

Examples

# make phrases from characters
phrase(c("a b", "c d e", "f"))

# from a dictionary
phrase(dictionary(list(catone = c("a b"), cattwo = "c d e", catthree = "f")))

# from a collocations object
(coll <- textstat_collocations(tokens("a b c a b d e b d a b")))
phrase(coll)
quanteda_options

Get or set package options for quanteda

Description

Get or set global options affecting functions across quanteda.

Usage

quanteda_options(..., reset = FALSE, initialize = FALSE)

Arguments

... options to be set, as key-value pair, same as options. This may be a list of valid key-value pairs, useful for setting a group of options at once (see examples).
reset logical; if TRUE, reset all quanteda options to their default values
initialize logical; if TRUE, reset only the quanteda options that are not already defined. Used for setting initial values when some have been defined previously, such as in `.Rprofile`.

Details

Currently available options are:

- verbose logical; if TRUE then use this as the default for all functions with a verbose argument
- threads integer; specifies the number of threads to use in parallelized functions
- print_dfm_max_ndoc integer; specifies the number of documents to display when using the defaults for printing a dfm
- print_dfm_max_nfeat integer; specifies the number of features to display when using the defaults for printing a dfm
- base_docname character; stem name for documents that are unnamed when a corpus, tokens, or dfm are created or when a dfm is converted from another object
- base_featname character; stem name for features that are unnamed when they are added, for whatever reason, to a dfm through an operation that adds features
- base_compname character; stem name for components that are created by matrix factorization
- language_stemmer character; language option for char_wordstem, tokens_wordstem, and dfm_wordstem

Value

When called using a key = value pair (where key can be a label or quoted character name)), the option is set and TRUE is returned invisibly.
When called with no arguments, a named list of the package options is returned.
When called with reset = TRUE as an argument, all arguments are options are reset to their default values, and TRUE is returned invisibly.
### Examples

```r
(opt <- quanteda_options())

quanteda_options(verbos = TRUE)
quanteda_options("verbos" = FALSE)
quanteda_options("threads")
quanteda_options(print_dfm_max_ndoc = 50L)
# reset to defaults
quanteda_options(reset = TRUE)
# reset to saved options
quanteda_options(opt)
```

---

**spacyr-methods**  
*Extensions for and from spacy_parse objects*

---

**Description**

These functions provide **quanteda** methods for **spacyr** objects, and also extend **spacy_parse** to work with **corpus** objects.

**Usage**

```r
## S3 method for class 'corpus'
spacy_parse(x, ...)
```

**Arguments**

- **x**: an object returned by `spacy_parse`, or (for `spacy_parse`) a **corpus** object
- **...**: unused except for `spacy_parse`, in which case it passes through extra arguments to that function

**Usage**

- `docnames(x)` returns the document names
- `ndoc(x)` returns the number of documents
- `ntoken(x,...)` returns the number of tokens by document
- `ntype(x,...)` returns the number of types (unique tokens) by document
- `spacy_parse(x,...)` is also defined for a **quanteda** corpus
### sparsity

*Compute the sparsity of a document-feature matrix*

#### Description

Return the proportion of sparseness of a document-feature matrix, equal to the proportion of cells that have zero counts.

#### Usage

\[
\text{sparsity}(x)
\]

#### Arguments

- **x**
  - the document-feature matrix

#### Examples

```r
dfmat <- dfm(data_corpus_inaugural)
sparsity(dfmat)
sparsity(dfm_trim(dfmat, min_termfreq = 5))
```
Description

textmodel_affinity implements the maximum likelihood supervised text scaling method described in Perry and Benoit (2017).

Usage

textmodel_affinity(
  x,
  y,
  exclude = NULL,
  smooth = 0.5,
  ref_smooth = 0.5,
  verbose = quanteda_options("verbose")
)

Arguments

x the dfm or bootstrap_dfm object on which the model will be fit. Does not need to contain only the training documents, since the index of these will be matched automatically.

y vector of training classes/scores associated with each document identified in data

exclude a set of words to exclude from the model

smooth a smoothing parameter for class affinities; defaults to 0.5 (Jeffreys prior). A plausible alternative would be 1.0 (Laplace prior).

ref_smooth a smoothing parameter for token distributions; defaults to 0.5

verbose logical; if TRUE print diagnostic information during fitting.

Author(s)

Patrick Perry and Kenneth Benoit

References


See Also

predict.textmodel_affinity for methods of applying a fitted textmodel_affinity model object to predict quantities from (other) documents.
textmodel_ca

Correspondence analysis of a document-feature matrix

Description

textmodel_ca implements correspondence analysis scaling on a dfm. The method is a fast/sparse version of function ca.

Usage

textmodel_ca(x, smooth = 0, nd = NA, sparse = FALSE, residual_floor = 0.1)

Arguments

- **x**: the dfm on which the model will be fit
- **smooth**: a smoothing parameter for word counts; defaults to zero.
- **nd**: Number of dimensions to be included in output; if NA (the default) then the maximum possible dimensions are included.
- **sparse**: retains the sparsity if set to TRUE; set it to TRUE if x (the dfm) is too big to be allocated after converting to dense
- **residual_floor**: specifies the threshold for the residual matrix for calculating the truncated svd. Larger value will reduce memory and time cost but might reduce accuracy; only applicable when sparse = TRUE

Details

svds in the RSpectra package is applied to enable the fast computation of the SVD.

Value

textmodel_ca() returns a fitted CA textmodel that is a special class of ca object.
Note
You may need to set sparse = TRUE and increase the value of residual_floor to ignore less important information and hence to reduce the memory cost when you have a very big dfm. If your attempt to fit the model fails due to the matrix being too large, this is probably because of the memory demands of computing the $V \times V$ residual matrix. To avoid this, consider increasing the value of residual_floor by 0.1, until the model can be fit.

Author(s)
Kenneth Benoit and Haiyan Wang

References

See Also
c_coef.textmodel_lsa, ca

Examples
dfmat <- dfm(data_corpus_irishbudget2010)
tmod <- textmodel_ca(dfmat)
summary(tmod)

textmodel_lsa Latent Semantic Analysis

Description
Fit the Latent Semantic Analysis scaling model to a dfm, which may be weighted (for instance using dfm_tfidf).

Usage
textmodel_lsa(x, nd = 10, margin = c("both", "documents", "features"))

Arguments
x the dfm on which the model will be fit
nd the number of dimensions to be included in output
margin margin to be smoothed by the SVD

Details
svds in the RSpectra package is applied to enable the fast computation of the SVD.
Note

The number of dimensions \( nd \) retained in LSA is an empirical issue. While a reduction in \( k \) can remove much of the noise, keeping too few dimensions or factors may lose important information.

Author(s)

Haiyan Wang and Kohei Watanabe

References


See Also

predict.textmodel_lsa, coef.textmodel_lsa

Examples

```r
dfmat <- dfm(data_corpus_irishbudget2010)
# create an LSA space and return its truncated representation in the low-rank space
tmod <- textmodel_lsa(dfmat[1:10, ])
head(tmod$docs)

# matrix in low_rank LSA space
tmod$matrix_low_rank[,1:5]

# fold queries into the space generated by dfmat[1:10,]
# and return its truncated versions of its representation in the new low-rank space
pred <- predict(tmod, newdata = dfmat[11:14, ])
pred$docs_newspace
```

---

**textmodel_nb**

*Naive Bayes classifier for texts*

**Description**

Fit a multinomial or Bernoulli Naive Bayes model, given a dfm and some training labels.
Usage

textmodel_nb(
  x,
  y,
  smooth = 1,
  prior = c("uniform", "docfreq", "termfreq"),
  distribution = c("multinomial", "Bernoulli")
)

Arguments

x       the dfm on which the model will be fit. Does not need to contain only the
         training documents.
y       vector of training labels associated with each document identified in train.
         (These will be converted to factors if not already factors.)
smooth  smoothing parameter for feature counts by class
prior    prior distribution on texts; one of "uniform", "docfreq", or "termfreq". See
         Prior Distributions below.
distribution  count model for text features, can be multinomial or Bernoulli. To fit a "bi-
                 nary multinomial" model, first convert the dfm to a binary matrix using dfm_weight(x, scheme = "boolean").

Value

textmodel_nb() returns a list consisting of the following (where \( I \) is the total number of doc-
ments, \( J \) is the total number of features, and \( k \) is the total number of training classes):

call       original function call
PwGc       \( k \times J \); probability of the word given the class (empirical likelihood)
Pc         \( k \)-length named numeric vector of class prior probabilities
PcGw       \( k \times J \); posterior class probability given the word
Pw         \( J \times 1 \); baseline probability of the word
x           the \( I \times J \) training dfm x
y           the \( I \)-length y training class vector
distribution the distribution argument
prior       the prior argument
smooth      the value of the smoothing parameter

Prior distributions

Prior distributions refer to the prior probabilities assigned to the training classes, and the choice of
prior distribution affects the calculation of the fitted probabilities. The default is uniform priors,
which sets the unconditional probability of observing the one class to be the same as observing any
other class.
"Document frequency" means that the class priors will be taken from the relative proportions of the class documents used in the training set. This approach is so common that it is assumed in many examples, such as the worked example from Manning, Raghavan, and Schütze (2008) below. It is not the default in quanteda, however, since there may be nothing informative in the relative numbers of documents used to train a classifier other than the relative availability of the documents. When training classes are balanced in their number of documents (usually advisable), however, then the empirically computed "docfreq" would be equivalent to "uniform" priors.

Setting prior to "termfreq" makes the priors equal to the proportions of total feature counts found in the grouped documents in each training class, so that the classes with the largest number of features are assigned the largest priors. If the total count of features in each training class was the same, then "uniform" and "termfreq" would be the same.

Author(s)

Kenneth Benoit

References


See Also

predict.textmodel_nb

Examples

```r
## Example from 13.1 of _An Introduction to Information Retrieval_  
res <- c(d1 = "Chinese Beijing Chinese",  
  d2 = "Chinese Chinese Shanghai",  
  d3 = "Chinese Macao",  
  d4 = "Tokyo Japan Chinese",  
  d5 = "Chinese Chinese Chinese Tokyo Japan")
trainingset <- dfm(res, tolower = FALSE)
trainingclass <- factor(c("Y", "Y", "Y", "N", NA), ordered = TRUE)

## replicate IIR p261 prediction for test set (document 5)  
(tmod1 <- textmodel_nb(trainingset, y = trainingclass, prior = "docfreq"))
summary(tmod1)
coef(tmod1)
predict(tmod1)

# contrast with other priors
predict(textmodel_nb(trainingset, y = trainingclass, prior = "uniform"))
predict(textmodel_nb(trainingset, y = trainingclass, prior = "termfreq"))
```
## replicate IIR p264 Bernoulli Naive Bayes

tmod2 <- textmodel_nb(trainingset, y = trainingclass, distribution = "Bernoulli",
                      prior = "docfreq")
predict(tmod2, newdata = trainingset[5, ])

---

# textmodel_wordfish

Wordfish text model

## Description


## Usage

textmodel_wordfish(
  x,
  dir = c(1, 2),
  priors = c(Inf, Inf, 3, 1),
  tol = c(1e-06, 1e-08),
  dispersion = c("poisson", "quasipoisson"),
  dispersion_level = c("feature", "overall"),
  dispersion_floor = 0,
  sparse = FALSE,
  abs_err = FALSE,
  svd_sparse = TRUE,
  residual_floor = 0.5
)

## Arguments

- **x**: the dfm on which the model will be fit
- **dir**: set global identification by specifying the indexes for a pair of documents such that $\hat{\theta}_{dir[1]} < \hat{\theta}_{dir[2]}$.
- **priors**: prior precisions for the estimated parameters $\alpha_i$, $\psi_j$, $\beta_j$, and $\theta_i$, where $i$ indexes documents and $j$ indexes features
- **tol**: tolerances for convergence. The first value is a convergence threshold for the log-posterior of the model, the second value is the tolerance in the difference in parameter values from the iterative conditional maximum likelihood (from conditionally estimating document-level, then feature-level parameters).
- **dispersion**: sets whether a quasi-Poisson quasi-likelihood should be used based on a single dispersion parameter ("poisson"), or quasi-Poisson ("quasipoisson")
- **dispersion_level**: sets the unit level for the dispersion parameter, options are "feature" for term-level variances, or "overall" for a single dispersion parameter
The returns match those of Will Lowe’s R implementation of wordfish (see the austin package), except that here we have renamed words to be features. (This return list may change.) We have also followed the practice begun with Slapin and Proksch’s early implementation of the model that used a regularization parameter of $se(\sigma) = 3$, through the third element in priors.

Value

An object of class textmodel_fitted_wordfish. This is a list containing:

- dir: global identification of the dimension
- theta: estimated document positions
- alpha: estimated document fixed effects
- beta: estimated feature marginal effects
- psi: estimated word fixed effects
- docs: document labels
- features: feature labels
- sigma: regularization parameter for betas in Poisson form
- ll: log likelihood at convergence
- se.theta: standard errors for theta-hats
- x: dfm to which the model was fit

Note

In the rare situation where a warning message of "The algorithm did not converge." shows up, removing some documents may work.

Author(s)

Benjamin Lauderdale, Haiyan Wang, and Kenneth Benoit
References


See Also

predict.textmodel_wordfish

Examples

(tmod1 <- textmodel_wordfish(data_dfm_lbgexample, dir = c(1,5)))
summary(tmod1, n = 10)
coef(tmod1)
predict(tmod1)
predict(tmod1, se.fit = TRUE)
predict(tmod1, interval = "confidence")

## Not run:
dfm <- dfm(data_corpus_irishbudget2010)
(tmod2 <- textmodel_wordfish(dfm, dir = c(6,5)))
(tmod3 <- textmodel_wordfish(dfm, dir = c(6,5), dispersion = "quasipoisson", dispersion_floor = 0))
(tmod4 <- textmodel_wordfish(dfm, dir = c(6,5), dispersion = "quasipoisson", dispersion_floor = .5))
plot(tmod3$phi, tmod4$phi, xlab = "Min underdispersion = 0", ylab = "Min underdispersion = .5", xlim = c(0, 1.0), ylim = c(0, 1.0))
plot(tmod3$phi, tmod4$phi, xlab = "Min underdispersion = 0", ylab = "Min underdispersion = .5", xlim = c(0, 1.0), ylim = c(0, 1.0), type = "n")
underdispersedTerms <- sample(which(tmod3$phi < 1.0), 5)
which(featnames(dfm) %in% names(topfeatures(dfm, 20)))
text(tmod3$phi, tmod4$phi, tmod3$features, cex = .8, xlim = c(0, 1.0), ylim = c(0, 1.0), col = "grey90")
text(tmod3$phi[underdispersedTerms], tmod4$phi[underdispersedTerms], tmod3$features[underdispersedTerms], cex = .8, xlim = c(0, 1.0), ylim = c(0, 1.0), col = "black")
if (requireNamespace("austin")) {
  tmod5 <- austin::wordfish(quanteda::as.wfm(dfm), dir = c(6,5))
cor(tmod1$theta, tmod5$theta)
}
## End(Not run)
textmodel_wordscores

Description

textmodel_wordscores implements Laver, Benoit and Garry’s (2003) "Wordscores" method for scaling texts on a single dimension, given a set of anchoring or reference texts whose values are set through reference scores. This scale can be fitted in the linear space (as per LBG 2003) or in the logit space (as per Beauchamp 2012). Estimates of virgin or unknown texts are obtained using the predict() method to score documents from a fitted textmodel_wordscores object.

Usage

textmodel_wordscores(x, y, scale = c("linear", "logit"), smooth = 0)

Arguments

x the dfm on which the model will be trained
y vector of training scores associated with each document in x
scale scale on which to score the words; "linear" for classic LBG linear posterior weighted word class differences, or "logit" for log posterior differences
smooth a smoothing parameter for word counts; defaults to zero to match the LBG (2003) method. See Value below for additional information on the behaviour of this argument.

Details

The textmodel_wordscores() function and the associated predict() method are designed to function in the same manner as predict.lm. coef() can also be used to extract the word coefficients from the fitted textmodel_wordscores object, and summary() will print a nice summary of the fitted object.

Value

A fitted textmodel_wordscores object. This object will contain a copy of the input data, but in its original form without any smoothing applied. Calling predict.textmodel_wordscores on this object without specifying a value for newdata, for instance, will predict on the unsmoothed object. This behaviour differs from versions of quanteda <= 1.2.

Author(s)

Kenneth Benoit

References


textplot_influence

Description
Plot the results of a fitted scaling model, from (e.g.) a predicted textmodel_affinity model.

Usage
textplot_influence(x, n = 30, ...)

Arguments
x the object output from 'influence()' run on the fitted or predicted scaling model object to be plotted
n the number of features whose influence will be plotted
... additional arguments passed to plot

Author(s)
Patrick Perry and Kenneth Benoit

See Also
textmodel_affinity
influence.predict.textmodel_affinity

Examples
tmod <- textmodel_affinity(data_dfm_lbgexample, y = c("L", NA, NA, NA, "R", NA))
pred <- predict(tmod)
textplot_influence(influence(pred))
textplot_keyness

*Plot word keyness*

**Description**

Plot the results of a "keyword" of features comparing their differential associations with a target and a reference group, after calculating keyness using `textstat_keyness`.

**Usage**

```r
textplot_keyness(
  x,
  show_reference = TRUE,
  show_legend = TRUE,
  n = 20L,
  min_count = 2L,
  margin = 0.05,
  color = c("darkblue", "gray"),
  labelcolor = "gray30",
  labelsize = 4,
  font = NULL
)
```

**Arguments**

- `x` a return object from `textstat_keyness`
- `show_reference` logical; if TRUE, show key reference features in addition to key target features
- `show_legend` logical; if TRUE, show legend
- `n` integer; number of features to plot
- `min_count` numeric; minimum total count of feature across the target and reference categories, for a feature to be included in the plot
- `margin` numeric; size of margin where feature labels are shown
- `color` character or integer; colors of bars for target and reference documents. `color` must have two elements when `show_reference = TRUE`. See `color`.
- `labelcolor` character; color of feature labels.
- `labelsize` numeric; size of feature labels and bars. See `size`.
- `font` character; font-family of texts. Use default font if `NULL`.

**Value**

a `ggplot2` object

**Author(s)**

Haiyan Wang and Kohei Watanabe
See Also

textstat_keyness

Examples

# compare Trump speeches to other Presidents by chi^2
dfmat1 <- data_corpus_inaugural %>%
corpus_subset(Year > 1980) %>%
dfm(groups = "President", remove = stopwords("english"), remove_punct = TRUE)
tstat1 <- textstat_keyness(dfmat1, target = "Trump")
textplot_keyness(tstat1, margin = 0.2, n = 10)

# compare contemporary Democrats v. Republicans
corp <- data_corpus_inaugural %>%
corpus_subset(Year > 1960)
docvars(corp, "party") <-
  ifelse(docvars(corp, "President") %in% c("Nixon", "Reagan", "Bush", "Trump"),
         "Republican", "Democrat")
dfmat2 <- dfm(corp, groups = "party", remove = stopwords("english"),
             remove_punct = TRUE)
tstat2 <- textstat_keyness(dfmat2, target = "Democrat", measure = "lr")
textplot_keyness(tstat2, color = c("blue", "red"), n = 10)
## S3 method for class 'fcm'

as.network(x, min_freq = 0.5, omit_isolated = TRUE, ...)

## S3 method for class 'fcm'

as.igraph(x, min_freq = 0.5, omit_isolated = TRUE, ...)

### Arguments

- **x**: a fcm or dfm object
- **min_freq**: a frequency count threshold or proportion for co-occurrence frequencies of features to be included.
- **omit_isolated**: if TRUE, features do not occur more frequent than min_freq will be omitted.
- **edge_color**: color of edges that connect vertices.
- **edge_alpha**: opacity of edges ranging from 0 to 1.0.
- **edge_size**: size of edges for most frequent co-occurrence The size of other edges are determined proportionally to the 99th percentile frequency instead of the maximum to reduce the impact of outliers.
- **vertex_color**: color of vertices.
- **vertex_size**: size of vertices
- **vertex_labelcolor**: color of texts. Defaults to the same as vertex_color. If NA is given, texts are not rendered.
- **vertex_labelfont**: font-family of texts. Use default font if NULL.
- **vertex_labelsize**: size of vertex labels in mm. Defaults to size 5. Supports both integer values and vector values.
- **offset**: if NULL, the distance between vertices and texts are determined automatically.
- **...**: additional arguments passed to network or graph_from_adjacency_matrix. Not used for as.igraph.

### Details

Currently the size of the network is limited to 1000, because of the computationally intensive nature of network formation for larger matrices. When the fcm is large, users should select features using fcm_select, set the threshold using min_freq, or implement own plotting function using as.network.

### Author(s)

Kohei Watanabe and Stefan Müller

### See Also

- fcm
- network
- graph_from_adjacency_matrix
Examples

set.seed(100)
toks <- corpus_subset(data_corpus_irishbudget2010) %>%
tokens(remove_punct = TRUE) %>%
tokens_tolower() %>%
tokens_remove(pattern = stopwords("english"), padding = FALSE)
fcmat <- fcm(toks, context = "window", tri = FALSE)
feat <- names(topfeatures(fcmat, 30))
fcm_select(fcmat, pattern = feat) %>%
textplot_network(min_freq = 0.5)
fcm_select(fcmat, pattern = feat) %>%
textplot_network(min_freq = 0.8)
fcm_select(fcmat, pattern = feat) %>%
textplot_network(min_freq = 0.8, vertex_labelcolor = rep(c("gray40", NA), 15))
fcm_select(fcmat, pattern = feat) %>%
textplot_network(min_freq = 0.8, vertex_labelsize = 10)
fcm_30 <- fcm_select(fcmat, pattern = feat)
textplot_network(fcm_30, vertex_labelsize = rowSums(fcm_30)/min(rowSums(fcm_30))
# Vector inputs to vertex_labelsize can be scaled if too small / large
# Vector inputs to vertex_labelsize can be scaled if too small / large

# as.igraph
if (requireNamespace("igraph", quietly = TRUE)) {
  txt <- c("a a a b b c", "a a c e", "a c e f g")
  mat <- fcm(txt)
  as.igraph(mat, min_freq = 1, omit_isolated = FALSE)
}

textplot_scale1d

Description

Plot the results of a fitted scaling model, from (e.g.) a predicted textmodel_wordscores model or a fitted textmodel_wordfish or textmodel_ca model. Either document or feature parameters may be plotted: an ideal point-style plot (estimated document position plus confidence interval on the x-axis, document labels on the y-axis) with optional renaming and sorting, or as a plot of estimated feature-level parameters (estimated feature positions on the x-axis, and a measure of relative frequency or influence on the y-axis, with feature names replacing plotting points with some being chosen by the user to be highlighted).

Usage

textplot_scale1d(
  x,
  margin = c("documents", "features"),
  doclabels = NULL,
  sort = TRUE,
  groups = NULL,
)
Arguments

x
the fitted or predicted scaling model object to be plotted

margin
"documents" to plot estimated document scores (the default) or "features" to plot estimated feature scores by a measure of relative frequency

doclabels
a vector of names for document; if left NULL (the default), docnames will be used

sort
if TRUE (the default), order points from low to high score. If a vector, order according to these values from low to high. Only applies when margin = "documents".

groups
either: a character vector containing the names of document variables to be used for grouping; or a factor or object that can be coerced into a factor equal in length or rows to the number of documents. See groups for details.

highlighted
a vector of feature names to draw attention to in a feature plot; only applies if margin = "features"

alpha
A number between 0 and 1 (default 0.5) representing the level of alpha transparency used to overplot feature names in a feature plot; only applies if margin = "features"

highlighted_color
color for highlighted terms in highlighted

Value

a ggplot2 object

Note

The groups argument only applies when margin = "documents".

Author(s)

Kenneth Benoit, Stefan Müller, and Adam Obeng

See Also

textmodel_wordfish, textmodel_wordscores, textmodel_ca

Examples

```r
## Not run:
dfmat <- dfm(data_corpus_irishbudget2010)

## wordscores
```
refscores <- c(rep(NA, 4), 1, -1, rep(NA, 8))
tmod1 <- textmodel_wordscores(dfmat, y = refscores, smooth = 1)
# plot estimated document positions
textplot_scale1d(predict(tmod1, se.fit = TRUE),
    groups = docvars(data_corpus_irishbudget2010, "party"))
# plot estimated word positions
textplot_scale1d(tmod1, highlighted = c("minister", "have", "our", "budget"))

## wordfish
tmod2 <- textmodel_wordfish(dfmat, dir = c(6,5))
# plot estimated document positions
textplot_scale1d(tmod2)
textplot_scale1d(tmod2, groups = docvars(data_corpus_irishbudget2010, "party"))
# plot estimated word positions
textplot_scale1d(tmod2, margin = "features",
    highlighted = c("government", "global", "children",
    "bank", "economy", "the", "citizenship",
    "productivity", "deficit"))

## correspondence analysis
tmod3 <- textmodel_ca(dfmat)
# plot estimated document positions
textplot_scale1d(tmod3, margin = "documents",
    groups = docvars(data_corpus_irishbudget2010, "party"))

## End(Not run)

textplot_wordcloud

Plot features as a wordcloud

Description

Plot a dfm object as a wordcloud, where the feature labels are plotted with their sizes proportional to their numerical values in the dfm. When comparison = TRUE, it plots comparison word clouds by document.

Usage

textplot_wordcloud(
    x,  
    min_size = 0.5,  
    max_size = 4,  
    min_count = 3,  
    max_words = 500,  
    color = "darkblue",  
    font = NULL,  
    adjust = 0,  
    rotation = 0.1,  
    random_order = FALSE,  
)
textplot_wordcloud

random_color = FALSE,
ordered_color = FALSE,
labelcolor = "gray20",
labelsizex = 1.5,
labeloffset = 0,
fixed_aspect = TRUE,
...
comparison = FALSE
)

Arguments

x a dfm object
min_size size of the smallest word
max_size size of the largest word
min_count words with frequency below min_count will not be plotted
max_words maximum number of words to be plotted. least frequent terms dropped.
color color of words from least to most frequent
font font-family of words and labels. Use default font if NULL.
adjust adjust sizes of words by a constant. Useful for non-English words for which R fails to obtain correct sizes.
rotation proportion of words with 90 degree rotation
random_order plot words in random order. If FALSE, they will be plotted in decreasing frequency.
random_color choose colors randomly from the colors. If FALSE, the color is chosen based on the frequency
ordered_color if TRUE, then colors are assigned to words in order.
labelcolor color of group labels. Only used when comparison=TRUE.
labelsizex size of group labels. Only used when comparison=TRUE.
labeloffset position of group labels. Only used when comparison=TRUE.
fixed_aspect if TRUE, the aspect ratio is fixed. Variable aspect ratio only supported if rotation = 0.
... additional parameters. Only used to make it compatible with wordcloud
comparison if TRUE, plot a wordcloud that compares documents in the same way as comparison.cloud

Details

The default is to plot the word cloud of all features, summed across documents. To produce word cloud plots for specific document or set of documents, you need to slice out the document(s) from the dfm object.

Comparison wordcloud plots may be plotted by setting comparison = TRUE, which plots a separate grouping for each document in the dfm. This means that you will need to slice out just a few documents from the dfm, or to create a dfm where the "documents" represent a subset or a grouping of documents by some document variable.
Author(s)

Kohei Watanabe, building on code from Ian Fellows’s wordcloud package.

Examples

```r
# plot the features (without stopwords) from Obama's inaugural addresses
set.seed(10)
dfmat1 <- dfm(corpus_subset(data_corpus_inaugural, President == "Obama"),
              remove = stopwords("english"), remove_punct = TRUE) %>%
dfm_trim(min_termfreq = 3)

# basic wordcloud
textplot_wordcloud(dfmat1)

# plot in colors with some additional options
textplot_wordcloud(dfmat1, rotation = 0.25,
                   color = rev(RColorBrewer::brewer.pal(10, "RdBu")))

# other display options
col <- sapply(seq(0.1, 1, 0.1), function(x) adjustcolor("#1F78B4", x))
textplot_wordcloud(dfmat1, adjust = 0.5, random_order = FALSE,
                   color = col, rotation = FALSE)

# comparison plot of Obama v. Trump
dfmat2 <- dfm(corpus_subset(data_corpus_inaugural, President %in% c("Obama", "Trump")),
              remove = stopwords("english"), remove_punct = TRUE, groups = "President") %>%
dfm_trim(min_termfreq = 3)
textplot_wordcloud(dfmat2, comparison = TRUE, max_words = 300,
                   color = c("blue", "red"))
```

textplot_xray

Plot the dispersion of key word(s)

Description

Plots a dispersion or "x-ray" plot of selected word pattern(s) across one or more texts. The format of the plot depends on the number of kwic class objects passed: if there is only one document, keywords are plotted one below the other. If there are multiple documents the documents are plotted one below the other, with keywords shown side-by-side. Given that this returns a ggplot2 object, you can modify the plot by adding ggplot2 layers (see example).

Usage

```r
textplot_xray(..., scale = c("absolute", "relative"), sort = FALSE)
```
Arguments

... any number of `kwic` class objects

scale whether to scale the token index axis by absolute position of the token in the document or by relative position. Defaults are absolute for single document and relative for multiple documents.

sort whether to sort the rows of a multiple document plot by document name

Value

a `ggplot2` object

Known Issues

These are known issues on which we are working to solve in future versions:

- `textplot_xray()` will not display the patterns correctly when these are multi-token sequences.
- For dictionaries with keys that have overlapping value matches to tokens in the text, only the first match will be used in the plot. The way around this is to produce one `kwic` per dictionary key, and send them as a list to `textplot_xray`.

Examples

```r
## Not run:
corp <- corpus_subset(data_corpus_inaugural, Year > 1970)
# compare multiple documents
textplot_xray(kwic(corp, pattern = "american"))
textplot_xray(kwic(corp, pattern = "american"), scale = "absolute")

# compare multiple terms across multiple documents
textplot_xray(kwic(corp, pattern = "america*"),
              kwic(corp, pattern = "people"))

# how to modify the ggplot with different options
library(ggplot2)
tplot <- textplot_xray(kwic(corp, pattern = "american"),
                        kwic(corp, pattern = "people"))
tplot + aes(color = keyword) + scale_color_manual(values = c('red', 'blue'))

# adjust the names of the document names
docnames(corp) <- apply(docvars(corp, c("Year", "President")), 1, paste, collapse = " ", ")
textplot_xray(kwic(corp, pattern = "america*"),
              kwic(corp, pattern = "people"))

## End(Not run)
```
texts

Get or assign corpus texts

Description
Get or replace the texts in a corpus, with grouping options. Works for plain character vectors too, if groups is a factor.

Usage
texts(x, groups = NULL, spacer = " ")
texts(x) <- value

## S3 method for class 'corpus'
as.character(x, ...)

Arguments
x a corpus or character object
groups either: a character vector containing the names of document variables to be used for grouping; or a factor or object that can be coerced into a factor equal in length or rows to the number of documents. See groups for details.
spacer when concatenating texts by using groups, this will be the spacing added between texts. (Default is two spaces.)
value character vector of the new texts
... unused

Details
as.character(x) where x is a corpus is equivalent to calling texts(x)

Value
For texts, a character vector of the texts in the corpus.
For texts <-, the corpus with the updated texts.
for texts <-, a corpus with the texts replaced by value
as.character(x) is equivalent to texts(x)

Note
The groups will be used for concatenating the texts based on shared values of groups, without any specified order of aggregation.
You are strongly encouraged as a good practice of text analysis workflow not to modify the substance of the texts in a corpus. Rather, this sort of processing is better performed through downstream operations. For instance, do not lowercase the texts in a corpus, or you will never be able to
recover the original case. Rather, apply tokens_tolower after applying tokens to a corpus, or use the option tolower = TRUE in dfm.

Examples

nchar(texts(corpus_subset(data_corpus_inaugural, Year < 1806)))

# grouping on a document variable
nchar(texts(corpus_subset(data_corpus_inaugural, Year < 1806), groups = "President"))

# grouping a character vector using a factor
nchar(data_char_ukimmig2010[1:5])
nchar(texts(data_corpus_inaugural[1:5],
    groups = as.factor(data_corpus_inaugural[1:5, "President"])))

BritCorpus <- corpus(c("We must prioritise honour in our neighbourhood.",
    "Aluminium is a valourous metal."))
texts(BritCorpus) <-
    stringi::stri_replace_all_regex(texts(BritCorpus),
    c("ise", "([nlb])our", "nium"),
    c("ize", "$1or", "num"),
    vectorize_all = FALSE)
texts(BritCorpus)
texts(BritCorpus)[2] <- "New text number 2."
texts(BritCorpus)

---

textstat_collocations  Identify and score multi-word expressions

Description

Identify and score multi-word expressions, or adjacent fixed-length collocations, from text.

Usage

textstat_collocations(
    x,
    method = "lambda",
    size = 2,
    min_count = 2,
    smoothing = 0.5,
    tolower = TRUE,
    ...
)

is.collocations(x)
Arguments

x  a character, corpus, or tokens object whose collocations will be scored. The tokens object should include punctuation, and if any words have been removed, these should have been removed with padding = TRUE. While identifying collocations for tokens objects is supported, you will get better results with character or corpus objects due to relatively imperfect detection of sentence boundaries from texts already tokenized.

method  association measure for detecting collocations. Currently this is limited to "lambda".

See Details.

size  integer; the length of the collocations to be scored

min_count  numeric; minimum frequency of collocations that will be scored

smoothing  numeric; a smoothing parameter added to the observed counts (default is 0.5)

tolower  logical; if TRUE, form collocations as lower-cased combinations

...  additional arguments passed to tokens, if x is not a tokens object already

Details

Documents are grouped for the purposes of scoring, but collocations will not span sentences. If x is a tokens object and some tokens have been removed, this should be done using tokens_remove(x, pattern, padding = TRUE) so that counts will still be accurate, but the pads will prevent those collocations from being scored.

The lambda computed for a size = K-word target multi-word expression the coefficient for the K-way interaction parameter in the saturated log-linear model fitted to the counts of the terms forming the set of eligible multi-word expressions. This is the same as the "lambda" computed in Blaheta and Johnson’s (2001), where all multi-word expressions are considered (rather than just verbs, as in that paper). The z is the Wald z-statistic computed as the quotient of lambda and the Wald statistic for lambda as described below.

In detail:

Consider a K-word target expression x, and let z be any K-word expression. Define a comparison function c(x, z) = (j1, ..., jK) = c such that the kth element of c is 1 if the kth word in z is equal to the kth word in x, and 0 otherwise. Let ci = (ji1, ..., jIK), i = 1, ..., 2K = M, be the possible values of c(x, z), with cM = (1, 1, ..., 1). Consider the set of c(x, zr) across all expressions zr in a corpus of text, and let ni, for i = 1, ..., M, denote the number of the c(x, zr) which equal ci, plus the smoothing constant smoothing. The ni are the counts in a 2^K contingency table whose dimensions are defined by the ci.

\lambda: The K-way interaction parameter in the saturated loglinear model fitted to the ni. It can be calculated as

\lambda = \sum_{i=1}^{M} (-1)^{K-b_i} * logn_i

where bi is the number of the elements of ci which are equal to 1.

Wald test z-statistic z is calculated as:

z = \frac{\lambda}{\sqrt{\sum_{i=1}^{M} n_i^{-1} (1/2)}}
Value

textstat_collocations returns a data.frame of collocations and their scores and statistics. This consists of the collocations, their counts, length, and λ and z statistics. When size is a vector, then count_nested counts the lower-order collocations that occur within a higher-order collocation (but this does not affect the statistics).

is.collocation returns TRUE if the object is of class collocations, FALSE otherwise.

Note

This function is under active development, with more measures to be added in the the next release of quanteda.

Author(s)

Kenneth Benoit, Jouni Kuha, Haiyan Wang, and Kohei Watanabe

References


Examples

corp <- data_corpus_inaugural[1:2]
head(cols <- textstat_collocations(corp, size = 2, min_count = 2), 10)
head(cols <- textstat_collocations(corp, size = 3, min_count = 2), 10)

# extracting multi-part proper nouns (capitalized terms)
toks1 <- tokens(data_corpus_inaugural)
toks2 <- tokens_remove(toks1, pattern = stopwords("english"), padding = TRUE)
toks3 <- tokens_select(toks2, pattern = "^([A-Z][a-z\-]{2,})", valuetype = "regex", case_insensitive = FALSE, padding = TRUE)
tstat <- textstat_collocations(toks3, size = 3, tolower = FALSE)
head(tstat, 10)

# vectorized size
txt <- c(".
" "a b c . . a b c . . c d e",
"a b . . a b . . a b . . a b",
"b c d . . b c . b c . . b c")
textstat_collocations(txt, size = 2:3)

textstat_entropy

Compute entropy of documents or features

Description

Compute entropy of documents or features
Usage

textstat_entropy(x, margin = c("documents", "features"), base = 2)

Arguments

x a dfm
margin character indicating which margin to compute entropy
base base for logarithm function

Examples

textstat_entropy(data_dfm_lbgexample)
textstat_entropy(data_dfm_lbgexample, "features")

textstat_frequency

Description

Tabulate feature frequencies

Produces counts and document frequencies summaries of the features in a dfm, optionally grouped by a docvars variable or other supplied grouping variable.

Usage

textstat_frequency(
  x,
  n = NULL,
  groups = NULL,
  ties_method = c("min", "average", "first", "random", "max", "dense"),
  ...
)

Arguments

x a dfm object
n (optional) integer specifying the top n features to be returned, within group if groups is specified
groups either: a character vector containing the names of document variables to be used for grouping; or a factor or object that can be coerced into a factor equal in length or rows to the number of documents. See groups for details.
ties_method character string specifying how ties are treated. See frank for details. Unlike that function, however, the default is "min", so that frequencies of 10, 10, 11 would be ranked 1, 1, 3.
... additional arguments passed to dfm_group. This can be useful in passing 'force = TRUE', for instance, if you are grouping a dfm that has been weighted.
Value

a data.frame containing the following variables:

feature (character) the feature
frequency count of the feature
rank rank of the feature, where 1 indicates the greatest frequency
docfreq document frequency of the feature, as a count (the number of documents in which this feature occurred at least once)
docfreq document frequency of the feature, as a count
group (only if groups is specified) the label of the group. If the features have been grouped, then all counts, ranks, and document frequencies are within group. If groups is not specified, the group column is omitted from the returned data.frame.

textstat_frequency returns a data.frame of features and their term and document frequencies within groups.

Examples

```r
set.seed(20)
dmat1 <- dfm(c("a a b b c d", "a d d d", "a a a"))
textstat_frequency(dmat1)
textstat_frequency(dmat1, groups = c("one", "two", "one"), ties_method = "first")
textstat_frequency(dmat1, groups = c("one", "two", "one"), ties_method = "dense")

dmat2 <- corpus_subset(data_corpus_inaugural, President == "Obama") %>%
df(remove_punct = TRUE, remove = stopwords("english"))
tstat1 <- textstat_frequency(dmat2)
head(tstat1, 10)

# plot 20 most frequent words
library("ggplot2")
ggplot(tstat1[1:20, ], aes(x = reorder(feature, frequency), y = frequency)) +
  geom_point() +
  coord_flip() +
  labs(x = NULL, y = "Frequency")

# plot relative frequencies by group
dmat3 <- data_corpus_inaugural %>%
corpus_subset(Year > 2000) %>%
df(remove = stopwords("english"), remove_punct = TRUE) %>%
df_group(groups = "President") %>%
df_weight(scheme = "prop")

# calculate relative frequency by president
tstat2 <- textstat_frequency(dmat3, n = 10, groups = "President")

# plot frequencies
ggplot(data = tstat2, aes(x = factor(nrow(tstat2):1), y = frequency)) +
  geom_point() +
```

textstat_keyness  

Calculate keyness statistics

Description

Calculate "keyness", a score for features that occur differentially across different categories. Here, the categories are defined by reference to a "target" document index in the \texttt{dfm}, with the reference group consisting of all other documents.

Usage

\begin{verbatim}
\texttt{textstat_keyness(x, target = 1L, measure = c("chi2", "exact", "lr", "pmi"), sort = TRUE, correction = c("default", "yates", "williams", "none"))}
\end{verbatim}

Arguments

- \texttt{x} \hspace{1cm} a \texttt{dfm} containing the features to be examined for keyness
- \texttt{target} \hspace{1cm} the document index (numeric, character or logical) identifying the document forming the "target" for computing keyness; all other documents’ feature frequencies will be combined for use as a reference
- \texttt{measure} \hspace{1cm} (signed) association measure to be used for computing keyness. Currently available: "chi2"; "exact" (Fisher’s exact test); "lr" for the likelihood ratio; "pmi" for pointwise mutual information.
- \texttt{sort} \hspace{1cm} logical; if \texttt{TRUE} sort features scored in descending order of the measure, otherwise leave in original feature order
- \texttt{correction} \hspace{1cm} if "default", Yates correction is applied to "chi2"; William’s correction is applied to "lr"; and no correction is applied for the "exact" and "pmi" measures. Specifying a value other than the default can be used to override the defaults, for instance to apply the Williams correction to the chi2 measure. Specifying a correction for the "exact" and "pmi" measures has no effect and produces a warning.
textstat_lexdiv

Value

a data.frame of computed statistics and associated p-values, where the features scored name each row, and the number of occurrences for both the target and reference groups. For measure = "chi2" this is the chi-squared value, signed positively if the observed value in the target exceeds its expected value; for measure = "exact" this is the estimate of the odds ratio; for measure = "lr" this is the likelihood ratio G2 statistic; for "pmi" this is the pointwise mutual information statistics.

textstat_keyness returns a data.frame of features and their keyness scores and frequency counts.

References


Examples

# compare pre- v. post-war terms using grouping
period <- ifelse(docvars(data_corpus_inaugural, "Year") < 1945, "pre-war", "post-war")
dfmat1 <- dfm(data_corpus_inaugural, groups = period)
head(dfmat1) # make sure 'post-war' is in the first row
head(tstat1 <- textstat_keyness(dfmat1), 10)
tail(tstat1, 10)

# compare pre- v. post-war terms using logical vector
dfmat2 <- dfm(data_corpus_inaugural)
head(textstat_keyness(dfmat2, docvars(data_corpus_inaugural, "Year") >= 1945), 10)

# compare Trump 2017 to other post-war preseidents
dfmat3 <- dfm(corpus_subset(data_corpus_inaugural, period == "post-war"))
head(textstat_keyness(dfmat3, target = "2017-Trump"), 10)

# using the likelihood ratio method
head(textstat_keyness(dfm_smooth(dfmat3), measure = "lr", target = "2017-Trump"), 10)

---

textstat_lexdiv

Calculate lexical diversity

Description

Calculate the lexical diversity of text(s).
Usage

textstat_lexdiv(
  remove_numbers = TRUE,
  remove_punct = TRUE,
  remove_symbols = TRUE,
  remove_hyphens = FALSE,
  log.base = 10,
  MATTR_window = 100L,
  MSTTR_segment = 100L,
  ...
)

Arguments

x an dfm or tokens input object for whose documents lexical diversity will be computed
measure a character vector defining the measure to compute
remove_numbers logical; if TRUE remove features or tokens that consist only of numerals (the Unicode "Number" [N] class)
remove_punct logical; if TRUE remove all features or tokens that consist only of the Unicode "Punctuation" [P] class)
remove_symbols logical; if TRUE remove all features or tokens that consist only of the Unicode "Punctuation" [S] class)
remove_hyphens logical; if TRUE split words that are connected by hyphenation and hyphenation-like characters in between words, e.g. "self-storage" becomes two features or tokens "self" and "storage". Default is FALSE to preserve such words as is, with the hyphens.
log.base a numeric value defining the base of the logarithm (for measures using logarithms)
MATTR_window a numeric value defining the size of the moving window for computation of the Moving-Average Type-Token Ratio (Covington & McFall, 2010)
MSTTR_segment a numeric value defining the size of the each segment for the computation of the Mean Segmental Type-Token Ratio (Johnson, 1944)
...
for passing arguments to other methods

Details

textstat_lexdiv calculates the lexical diversity of documents using a variety of indices.
In the following formulas, \( N \) refers to the total number of tokens, \( V \) to the number of types, and \( f_v(i, N) \) to the numbers of types occurring \( i \) times in a sample of length \( N \).

"TTR": The ordinary Type-Token Ratio: 

\[
TTR = \frac{V}{N}
\]
"C": Herdan’s C (Herdan, 1960, as cited in Tweedie & Baayen, 1998; sometimes referred to as \( \log TTR \)):

\[
C = \frac{\log V}{\log N}
\]

"R": Guiraud’s \( \text{Root TTR} \) (Guiraud, 1954, as cited in Tweedie & Baayen, 1998):

\[
R = \frac{V}{\sqrt{N}}
\]

"CTTR": Carroll’s \( \text{Corrected TTR} \):

\[
CTTR = \frac{V}{\sqrt{2N}}
\]

"U": Dugast’s \( \text{Uber Index} \) (Dugast, 1978, as cited in Tweedie & Baayen, 1998):

\[
U = \frac{(\log N)^2}{\log N - \log V}
\]

"S": Summer’s index:

\[
S = \frac{\log \log V}{\log \log N}
\]

"K": Yule’s \( K \) (Yule, 1944, as presented in Tweedie & Baayen, 1998, Eq. 16) is calculated by:

\[
K = 10^4 \times \left[ \frac{1}{N} + \sum_{i=1}^{V} f_v(i, N) \left( \frac{i}{N} \right)^2 \right]
\]

"I": Yule’s \( I \) (Yule, 1944) is calculated by:

\[
I = \frac{V^2}{M_2 - V}
\]

\[
M_2 = \sum_{i=1}^{V} i^2 \cdot f_v(i, N)
\]

"D": Simpson’s \( D \) (Simpson 1949, as presented in Tweedie & Baayen, 1998, Eq. 17) is calculated by:

\[
D = \sum_{i=1}^{V} f_v(i, N) \frac{i}{N} \frac{i - 1}{N - 1}
\]

"Vm": Herdan’s \( V_m \) (Herdan 1955, as presented in Tweedie & Baayen, 1998, Eq. 18) is calculated by:

\[
V_m = \sqrt{\sum_{i=1}^{V} f_v(i, N)(i/N)^2 - \frac{i}{V}}
\]

"Maas": Maas’ indices \( (a, \log V_0 & \log_e V_0) \):

\[
a^2 = \frac{\log N - \log V}{\log N^2}
\]
\[ \log V_0 = \frac{\log V}{\sqrt{1 - \frac{\log V^2}{\log N}}} \]

The measure was derived from a formula by Mueller (1969, as cited in Maas, 1972). \( \log_e V_0 \) is equivalent to \( \log V_0 \), only with \( e \) as the base for the logarithms. Also calculated are \( a, \log V_0 \) (both not the same as before) and \( V' \) as measures of relative vocabulary growth while the text progresses. To calculate these measures, the first half of the text and the full text will be examined (see Maas, 1972, p. 67 ff. for details). Note: for the current method (for a dfm) there is no computation on separate halves of the text.

"MATTR": The Moving-Average Type-Token Ratio (Covington & McFall, 2010) calculates TTRs for a moving window of tokens from the first to the last token, computing a TTR for each window. The MATTR is the mean of the TTRs of each window.

"MSTTR": Mean Segmental Type-Token Ratio (sometimes referred to as Split TTR) splits the tokens into segments of the given size, TTR for each segment is calculated and the mean of these values returned. When this value is < 1.0, it splits the tokens into equal, non-overlapping sections of that size. When this value is > 1, it defines the segments as windows of that size. Tokens at the end which do not make a full segment are ignored.

**Value**

A data.frame of documents and their lexical diversity scores.

**Author(s)**

Kenneth Benoit and Jiong Wei Lua. Many of the formulas have been reimplemented from functions written by Meik Michalke in the koRpus package.

**References**


Examples

txt <- c("Anyway, like I was sayin', shrimp is the fruit of the sea. You can barbecue it, boil it, broil it, bake it, saute it.", "There's shrimp-kabobs, shrimp creole, shrimp gumbo. Pan fried, deep fried, stir-fried. There's pineapple shrimp, lemon shrimp, coconut shrimp, pepper shrimp, shrimp soup, shrimp stew, shrimp salad, shrimp and potatoes, shrimp burger, shrimp sandwich.")
tokens(txt) %>%
textstat_lexdiv(measure = c("TTR", "CTTR", "K"))
dfm(txt) %>%
textstat_lexdiv(measure = c("TTR", "CTTR", "K"))
toks <- tokens(corpus_subset(data_corpus_inaugural, Year > 2000))
textstat_lexdiv(toks, c("CTTR", "TTR", "MATTR"), MATTR_window = 100)

Description

Calculate the readability of text(s) using one of a variety of computed indexes.

Usage

textstat_readability(
  x, 
  measure = "Flesch", 
  remove_hyphens = TRUE,
  min_sentence_length = 1,
  max_sentence_length = 10000,
  intermediate = FALSE,
  ...
)

Arguments

x a character or corpus object containing the texts
measure character vector defining the readability measure to calculate. Matches are case-insensitive. See other valid measures under Details.
remove_hyphens if TRUE, treat constituent words in hyphenated as separate terms, for purposes of computing word lengths, e.g. "decision-making" as two terms of lengths 8 and 6 characters respectively, rather than as a single word of 15 characters
min_sentence_length, max_sentence_length set the minimum and maximum sentence lengths (in tokens, excluding punctuation) to include in the computation of readability. This makes it easy to exclude
"sentences" that may not really be sentences, such as section titles, table elements, and other cruft that might be in the texts following conversion. For finer-grained control, consider filtering sentences prior first, including through pattern-matching, using `corpus_trim`

intermediate if TRUE, include intermediate quantities in the output

... not used

Details

The following readability formulas have been implemented, where

- \( N_w = n_w \) = number of words
- \( N_c = n_c \) = number of characters
- \( N_{st} = n_{st} \) = number of sentences
- \( N_{sy} = n_{sy} \) = number of syllables
- \( N_{wf} = n_{wf} \) = number of words matching the Dale-Chall List of 3000 "familiar words"
- \( \text{ASL} = \text{Average Sentence Length: number of words / number of sentences} \)
- \( \text{AWL} = \text{Average Word Length: number of characters / number of words} \)
- \( \text{AFW} = \text{Average Familiar Words: count of words matching the Dale-Chall list of 3000 "familiar words" / number of all words} \)
- \( N_{wd} = n_{wd} \) = number of "difficult" words not matching the Dale-Chall list of "familiar" words

"ARI": Automated Readability Index (Senter and Smith 1967)

\[
0.5 \text{ASL} + 4.71 \text{AWL} - 21.34
\]

"ARI.Simple": A simplified version of Senter and Smith’s (1967) Automated Readability Index.

\[
\text{ASL} + 9 \text{AWL}
\]

"Bormuth.MC": Bormuth’s (1969) Mean Cloze Formula.

\[
0.886593 - 0.03640 \times \text{AWL} + 0.161911 \times \text{AFW} - 0.21401 \times \text{ASL} - 0.000577 \times \text{ASL}^2 - 0.000005 \times \text{ASL}^3
\]

"Bormuth.GP": Bormuth’s (1969) Grade Placement score.

\[
4.275 + 12.881M - 34.934M^2 + 20.388M^3 + 26.194CCS - 2.046CCS^2 - 11.767CCS^3 - 42.285(M \times CCS) + 97.620(CC1) - 59.538(CC2)
\]

where \( M \) is the Bormuth Mean Cloze Formula as in "Bormuth" above, and CCS is the Cloze Criterion Score (Bormuth, 1968).


\[
1.29 \times \frac{100 \times n_{w_{sy}=1}}{n_w} - 38.45
\]

where \( n_{w_{sy}=1} = N_{sy1} = \) the number of one-syllable words. The scaling by 100 in this and the other Coleman-derived measures arises because the Coleman measures are calculated on a per 100 words basis.

\[
1.16 \times \frac{100 \times n_{wsp}}{Nw + 1.48 \times \frac{100 \times n_{st}}{n_w}} - 37.95
\]


\[
141.8401 - 0.214590 \times 100 \times \text{AWL} + 1.079812 \times \frac{n_{st} \times 100}{n_w}
\]

"Coleman.Liau.grade": Coleman-Liau Grade Level (Coleman and Liau 1975).

\[
-27.4004 \times \text{Coleman.Liau.ECP} \times 100 + 23.06395
\]


\[
5.88 \times \text{AWL} + 29.6 \times \frac{n_{st}}{n_w} - 15.8
\]


\[
64 - (0.95 \times 100 \times \frac{n_{wd}}{n_w}) - (0.69 \times \text{ASL})
\]


\[
0.1579 \times 100 \times \frac{n_{wd}}{n_w} + 0.0496 \times \text{ASL} + 3.6365
\]

The additional constant 3.6365 is only added if \((N_{wd} / N_w) > 0.05\).


\[
0.1155 \times 100 \times \frac{n_{wd}}{n_w} + (0.0596 \times \text{ASL}) + 3.2672
\]


\[
(1.0364 \times \frac{n_c}{n_{blank}}) + (0.0194 \times \frac{n_c}{n_{st}}) - 0.6059
\]

where \(n_{blank} = N_{blank} = \) the number of blanks.


\[
131.059 - (10.364 \times \frac{n_c}{n_{blank}}) + (0.0194 \times \frac{n_c}{n_{st}})
\]

where \(n_{blank} = N_{blank} = \) the number of blanks.

"Dickes.Steiwer": Dickes-Steiwer Index (Dicks and Steiwer 1977).

\[
235.95993 - (7.3021 \times \text{AWL}) - (12.56438 \times \text{ASL}) - (50.03293 \times \text{TTR})
\]

where TTR is the Type-Token Ratio (see textstat_lexdiv).
"DRP": Degrees of Reading Power.

\[ (1 - Bormuth.MC) \times 100 \]

where Bormuth.MC refers to Bormuth’s (1969) Mean Cloze Formula (documented above)

"ELF": Easy Listening Formula (Fang 1966):

\[ \frac{n_{wsy\geq2}}{n_{st}} \]

where \( n_{wsy\geq2} = N\text{wmin2sy} \) = the number of words with 2 syllables or more.

"Farr.Jenkins.Paterson": Farr-Jenkins-Paterson’s Simplification of Flesch’s Reading Ease Score (Farr, Jenkins and Paterson 1951).

\[-31.517 - (1.015 \times ASL) + (1.599 \times \frac{n_{wsy=1}}{n_w})\]

where \( n_{wsy=1} = N\text{wsy1} \) = the number of one-syllable words.

"Flesch": Flesch’s Reading Ease Score (Flesch 1948).

\[ 206.835 - (1.015 \times ASL) - (84.6 \times \frac{n_{sy}}{n_w}) \]

"Flesch.PSK": The Powers-Sumner-Kearl’s Variation of Flesch Reading Ease Score (Powers, Sumner and Kearl, 1958).

\[ (0.0778 \times ASL) + (4.55 \times \frac{n_{sy}}{n_w}) - 2.2029 \]

"Flesch.Kincaid": Flesch-Kincaid Readability Score (Flesch and Kincaid 1975).

\[ 0.39 \times ASL + 11.8 \times \frac{n_{sy}}{n_w} - 15.59 \]

"FOG": Gunning’s Fog Index (Gunning 1952).

\[ 0.4 \times (ASL + 100 \times \frac{n_{wsy\geq3}}{n_w}) \]

where \( n_{wsy\geq3} = N\text{wmin3sy} \) = the number of words with 3-syllables or more. The scaling by 100 arises because the original FOG index is based on just a sample of 100 words.

"FOG.PSK": The Powers-Sumner-Kearl Variation of Gunning’s Fog Index (Powers, Sumner and Kearl, 1958).

\[ 3.0680 \times (0.0877 \times ASL) + (0.0984 \times 100 \times \frac{n_{wsy\geq3}}{n_w}) \]

where \( n_{wsy\geq3} = N\text{wmin3sy} \) = the number of words with 3-syllables or more. The scaling by 100 arises because the original FOG index is based on just a sample of 100 words.

"FOG.NRI": The Navy’s Adaptation of Gunning’s Fog Index (Kincaid, Fishburne, Rogers and Chissom 1975).

\[ \left( \frac{(n_{wsy<3} + 3 \times n_{wsy=3})}{100 \times \frac{n_{w}}{Nw}} - 3 \right) / 2 \]

where \( n_{wsy<3} = N\text{wless3sy} \) = the number of words with less than 3 syllables, and \( n_{wsy=3} = N\text{w3sy} \) = the number of 3-syllable words. The scaling by 100 arises because the original FOG index is based on just a sample of 100 words.
"FORCAST": FORCAST (Simplified Version of FORCAST.RGL) (Caylor and Sticht 1973).

\[
20 - \frac{n_{wsy=1} \times 150}{(n_w \times 10)}
\]

where \(n_{wsy=1} = N_{wsy1}\) = the number of one-syllable words. The scaling by 150 arises because the original FORCAST index is based on just a sample of 150 words.

"FORCAST.RGL": FORCAST.RGL (Caylor and Sticht 1973).

\[
20.43 - 0.11 \times \frac{n_{wsy=1} \times 150}{(n_w \times 10)}
\]

where \(n_{wsy=1} = N_{wsy1}\) = the number of one-syllable words. The scaling by 150 arises because the original FORCAST index is based on just a sample of 150 words.

"Fucks": Fucks’ (1955) Stilcharakteristik (Style Characteristic).

\[
AWL \ast ASL
\]

"Linsear.Write": Linsear Write (Klare 1975).

\[
\frac{[(100 - \frac{100 \times n_{wsy<3}}{n_w}) + (3 \times \frac{100 \times n_{wsy>=3}}{n_w})]}{(100 \times \frac{n_{stn_w}}{n_w})}
\]

where \(n_{wsy<3} = N_{wless3sy}\) = the number of words with less than 3 syllables, and \(n_{wsy>=3} = N_{wmin3sy}\) = the number of words with 3-syllables or more. The scaling by 100 arises because the original Linsear.Write measure is based on just a sample of 100 words.

"LIW": Björnsson’s (1968) Läsbarhetsindex (For Swedish Texts).

\[
ASL + \frac{100 \times n_{wsy>=7}}{n_w}
\]

where \(n_{wsy>=7} = N_{wmin7sy}\) = the number of words with 7-syllables or more. The scaling by 100 arises because the Läsbarhetsindex index is based on just a sample of 100 words.

"nWS": Neue Wiener Sachtextformeln 1 (Bamberger and Vanecek 1984).

\[
19.35 \times \frac{n_{wsy>=3}}{n_w} + 0.1672 \times ASL + 12.97 \times \frac{b_{wchar>=6}}{n_w} - 3.27 \times \frac{n_{wsy=1}}{n_w} - 0.875
\]

where \(n_{wsy>=3} = N_{wmin3sy}\) = the number of words with 3 syllables or more, \(n_{wchar>=6} = N_{wmin6char}\) = the number of words with 6 characters or more, and \(n_{wsy=1} = N_{wsy1}\) = the number of one-syllable words.

"nWS.2": Neue Wiener Sachtextformeln 2 (Bamberger and Vanecek 1984).

\[
20.07 \times \frac{n_{wsy>=3}}{n_w} + 0.1682 \times ASL + 13.73 \times \frac{n_{wchar>=6}}{n_w} - 2.779
\]

where \(n_{wsy>=3} = N_{wmin3sy}\) = the number of words with 3 syllables or more, and \(n_{wchar>=6} = N_{wmin6char}\) = the number of words with 6 characters or more.
"nWS.3": Neue Wiener Sachtextformeln 3 (Bamberger and Vanecek 1984).

\[
29.63 \times \frac{n_{wsy>3}}{n_w} + 0.1905 \times ASL - 1.1144
\]

where \( n_{wsy>3} = N_{w\text{min3sy}} \) = the number of words with 3 syllables or more.

"nWS.4": Neue Wiener Sachtextformeln 4 (Bamberger and Vanecek 1984).

\[
27.44 \times \frac{n_{wsy>3}}{n_w} + 0.2656 \times ASL - 1.693
\]

where \( n_{wsy>3} = N_{w\text{min3sy}} \) = the number of words with 3 syllables or more.

"RIX": Anderson’s (1983) Readability Index.

\[
\frac{n_{wsy>7}}{n_{st}}
\]

where \( n_{wsy>7} = N_{w\text{min7sy}} \) = the number of words with 7-syllables or more.

"Scrabble": Scrabble Measure.

\[
\text{MeanScrabbleLetterValuesofAllWords}
\]

. Scrabble values are for English. There is no reference for this, as we created it experimentally. It’s not part of any accepted readability index!

"SMOG": Simple Measure of Gobbledygook (SMOG) (McLaughlin 1969).

\[
1.043 \times \sqrt{n_{wsy>3}} \times \frac{30}{n_{st}} + 3.1291
\]

where \( n_{wsy>3} = N_{w\text{min3sy}} \) = the number of words with 3 syllables or more. This measure is regression equation D in McLaughlin’s original paper.

"SMOG.C": SMOG (Regression Equation C) (McLaughlin’s 1969)

\[
0.9986 \times \sqrt{N_{w\text{min3sy}} \times \frac{30}{n_{st}}} + 5 + 2.8795
\]

where \( n_{wsy>3} = N_{w\text{min3sy}} \) = the number of words with 3 syllables or more. This measure is regression equation C in McLaughlin’s original paper.

"SMOG.simple": Simplified Version of McLaughlin’s (1969) SMOG Measure.

\[
\sqrt{N_{w\text{min3sy}} \times \frac{30}{n_{st}}} + 3
\]

"SMOG.de": Adaptation of McLaughlin’s (1969) SMOG Measure for German Texts.

\[
\sqrt{N_{w\text{min3sy}} \times \frac{30}{n_{st}}} - 2
\]

"Spache": Spache’s (1952) Readability Measure.

\[
0.121 \times ASL + 0.082 \times \frac{n_{w\text{notinspache}}}{n_w} + 0.659
\]

where \( n_{w\text{notinspache}} = N_{w\text{notinspache}} \) = number of unique words not in the Spache word list.
"Spache.old": Spache’s (1952) Readability Measure (Old).

\[ 0.141 \times ASL + 0.086 \times \frac{n_{\text{wnotinspache}}}{n_w} + 0.839 \]

where \( n_{\text{wnotinspache}} = N_{\text{wnotinspache}} \) = number of unique words not in the Spache word list.

"Strain": Strain Index (Solomon 2006).

\[ n_{\text{sy}} \frac{n_{\text{st}}}{3} / 10 \]

The scaling by 3 arises because the original Strain index is based on just the first 3 sentences.


\[ 224.6814 - (79.8304 \times AWL) - (12.24032 \times ASL) - (1.29257 \times 100 \times \frac{n_{\text{prep}}}{n_w}) \]

where \( n_{\text{prep}} = N_{\text{prep}} \) = the number of prepositions. The scaling by 100 arises because the original Tränkle & Bailer index is based on just a sample of 100 words.


\[ Tr\text{nkle.Bailer2} = 234.1063 - (96.11069 \times AWL) - (2.05444 \times 100 \times \frac{n_{\text{prep}}}{n_w}) - (1.02805 \times 100 \times \frac{n_{\text{conj}}}{n_w}) \]

where \( n_{\text{prep}} = N_{\text{prep}} \) = the number of prepositions, \( n_{\text{conj}} = N_{\text{conj}} \) = the number of conjunctions. The scaling by 100 arises because the original Tränkle & Bailer index is based on just a sample of 100 words.


\[ ASL \times 10 \times \frac{n_{\text{w} sy > 2}}{n_{\text{words}}} \]

where \( n_{\text{w} sy > 2} = N_{\text{wmin2sy}} \) = the number of words with 2 syllables or more.

"meanSentenceLength": Average Sentence Length (ASL).

\[ \frac{n_{w}}{n_{st}} \]

"meanWordSyllables": Average Word Syllables (AWL).

\[ \frac{n_{sy}}{n_{w}} \]

Value

textstat_readability returns a data.frame of documents and their readability scores.

Author(s)

Kenneth Benoit, re-engineered from Meik Michalke’s koRpus package.
References


*Nimaldasan is the pen name of N. Watson Solomon, Assistant Professor of Journalism, School of Media Studies, SRM University, India.*

### Examples

```r
txt <- c(doc1 = "Readability zero one. Ten, Eleven.",
          doc2 = "The cat in a dilapidated tophat.")
textstat_readability(txt, measure = "Flesch")
textstat_readability(txt, measure = c("FOG", "FOG.PSK", "FOG.NRI"))
textstat_readability(data_corpus_inaugural[48:58],
                      measure = c("Flesch.Kincaid", "Dale.Chall.old"))
```

### textstat_simil

**Similarity and distance computation between documents or features**

These functions compute matrixes of distances and similarities between documents or features from a dfm and return a matrix of similarities or distances in a sparse format. These methods are fast and robust because they operate directly on the sparse dfm objects. The output can easily be coerced to an ordinary matrix, a data.frame of pairwise comparisons, or a dist format.

### Usage

```r
textstat_simil(
  x,
  y = NULL,
  selection = NULL,
  margin = c("documents", "features"),
  method = c("correlation", "cosine", "jaccard", "ejaccard", "dice", "edice", "hamman",
             "simple matching"),
  min_simil = NULL,
  ...
)
```
textstat_dist(
  x,
  y = NULL,
  selection = NULL,
  margin = c("documents", "features"),
  method = c("euclidean", "manhattan", "maximum", "canberra", "minkowski"),
  p = 2,
  ...
)

## S3 method for class 'textstat_proxy'
as.list(x, sorted = TRUE, n = NULL, diag = FALSE, ...)

## S3 method for class 'textstat_proxy'
as.data.frame(  
  x,
  row.names = NULL,
  optional = FALSE,
  diag = FALSE,
  upper = FALSE,
  ...
)

Arguments

  x, y  a dfm objects; y is an optional target matrix matching x in the margin on which
  the similarity or distance will be computed.

  selection (deprecated - use y instead).

  margin identifies the margin of the dfm on which similarity or difference will be com-
  puted: "documents" for documents or "features" for word/term features.

  method character; the method identifying the similarity or distance measure to be used;
  see Details.

  min_simil numeric; a threshold for the similarity values below which similarity values will
  not be returned

  ... unused

  p The power of the Minkowski distance.

  sorted sort results in descending order if TRUE

  n the top n highest-ranking items will be returned. If n is NULL, return all items.

  diag logical; if FALSE, exclude the item's comparison with itself

  row.names NULL or a character vector giving the row names for the data frame. Missing
  values are not allowed.

  optional logical. If TRUE, setting row names and converting column names (to syntac-
  tic names: see make.names) is optional. Note that all of R's base package
  as.data.frame() methods use optional only for column names treatment,
  basically with the meaning of data.frame(*,check.names = !optional). See
  also the make.names argument of the matrix method.

  upper logical; if TRUE, return pairs as both (A, B) and (B, A)
Details

textstat_simil options are: "correlation" (default), "cosine", "jaccard", "ejaccard", "dice", "edice", "simple matching", and "hamman".
textstat_dist options are: "euclidean" (default), "manhattan", "maximum", "canberra", and "minkowski".

Value

A sparse matrix from the Matrix package that will be symmetric unless y is specified.
These can be transformed easily into a list format using as.list(), which returns a list for each unique element of the second of the pairs, as.dist to be transformed into a dist object, or as.matrix to convert it into an ordinary matrix.

as.data.list for a textstat_simil or textstat_dist object returns a list equal in length to the columns of the simil or dist object, with the rows and their values as named elements. By default, this list excludes same-time pairs (when diag = FALSE) and sorts the values in descending order (when sorted = TRUE).

as.data.frame for a textstat_simil or textstat_dist object returns a data.frame of pairwise combinations and the and their similarity or distance value.

Note
If you want to compute similarity on a "normalized" dfm object (controlling for variable document lengths, for methods such as correlation for which different document lengths matter), then wrap the input dfm in dfm_weight(x,"prop").

See Also

as.dist

Examples

# similarities for documents
dfmat <- dfm(corpus_subset(data_corpus_inaugural, Year > 2000),
            remove_punct = TRUE, remove = stopwords("english"))
(tstat1 <- textstat_simil(dfmat, method = "cosine", margin = "documents"))
as.matrix(tstat1)
as.list(tstat1)
as.list(tstat1, diag = TRUE)

# min_simil
(tstat2 <- textstat_simil(dfmat, method = "cosine", margin = "documents", min_simil = 0.6))
as.matrix(tstat2)

# similarities for for specific documents

textstat_simil(dfmat, dfmat["2017-Trump", ], margin = "documents")
textstat_simil(dfmat, dfmat["2017-Trump", ], method = "cosine", margin = "documents")
textstat_simil(dfmat, dfmat["2009-Obama", "2013-Obama"], ], margin = "documents")

# compute some term similarities
tokens <- textstat_simil(dfmat, dfmat[, c("fair", "health", "terror")], method = "cosine", margin = "features")
head(as.matrix(tokens), 10)
as.list(tokens, n = 6)

# distances for documents
(tstat4 <- textstat_dist(dfmat, margin = "documents"))
as.matrix(tstat4)
as.list(tstat4)
as.dist(tstat4)

# distances for specific documents
textstat_dist(dfmat, dfmat["2017-Trump", ], margin = "documents")
(tstat5 <- textstat_dist(dfmat, dfmat[c("2009-Obama", "2013-Obama"), ], margin = "documents"))
as.matrix(tstat5)
as.list(tstat5)

## Not run:
# plot a dendrogram after converting the object into distances
plot(hclust(as.dist(tstat4)))
## End(Not run)

---

tokens

Tokenize a set of texts

description

Tokenize the texts from a character vector or from a corpus.

usage

tokens(
  x,
  what = c("word", "sentence", "character", "fastestword", "fasterword"),
  remove_numbers = FALSE,
  remove_punct = FALSE,
  remove_symbols = remove_punct,
  remove_separators = TRUE,
  remove_twitter = FALSE,
  remove_hyphens = FALSE,
  remove_url = FALSE,
  ngrams = 1L,
  skip = 0L,
  concatenator = ",",
  verbose = quantmla_options("verbose"),
  include_docvars = TRUE,
  ...
)
Arguments

x
a character, corpus, or tokens object to be tokenized

what
the unit for splitting the text, available alternatives are:

"word" (recommended default) smartest, but slowest, word tokenization method; see stringi-search-boundaries for details.

"fasterword" dumber, but faster, word tokenization method, uses stri_split_charclass(x,"[\p{Z}\p{C}]+")

"fastestword" dumbest, but fastest, word tokenization method, calls stri_split_fixed(x,"
"")

"character" tokenization into individual characters

"sentence" sentence segmenter, smart enough to handle some exceptions in English such as "Prof. Plum killed Mrs. Peacock." (but far from perfect).

remove_numbers
logical; if TRUE remove tokens that consist only of numbers, but not words that start with digits, e.g. 2day

remove_punct
logical; if TRUE remove all characters in the Unicode "Punctuation" [P] class

remove_symbols
logical; if TRUE remove all characters in the Unicode "Symbol" [S] class

remove_separators
logical; if TRUE remove separators and separator characters (Unicode "Separator" [Z] and "Control [C]" categories). Only applicable for what = "character" (when you probably want it to be FALSE) and for what = "word" (when you probably want it to be TRUE).

remove_twitter
logical; if TRUE remove Twitter characters @ and #; set to TRUE if you wish to eliminate these. Note that this will always be set to FALSE if remove_punct = FALSE.

remove_hyphens
logical; if TRUE split words that are connected by hyphenation and hyphenation-like characters in between words, e.g. "self-storage" becomes c("self","storage"). Default is FALSE to preserve such words as is, with the hyphens. Only applies if what = "word" or what = "fasterword".

remove_url
logical; if TRUE find and eliminate URLs beginning with http(s) – see section "Dealing with URLs".

ngrams
integer vector of the n for n-grams, defaulting to 1 (unigrams). For bigrams, for instance, use 2; for bigrams and unigrams, use 1:2. You can even include irregular sequences such as 2:3 for bigrams and trigrams only. See tokens_ngrams.

skip
integer vector specifying the skips for skip-grams, default is 0 for only immediately neighbouring words. Only applies if ngrams is different from the default of 1. See tokens_skipgrams.

concatenator
character to use in concatenating n-grams, default is ".", which is recommended since this is included in the regular expression and Unicode definitions of "word" characters

verbose
if TRUE, print timing messages to the console; off by default

include_docvars
if TRUE, pass docvars and metadoc fields through to the tokens object. Only applies when tokenizing corpus objects.

... additional arguments not used
Details

The tokenizer is designed to be fast and flexible as well as to handle Unicode correctly. Most of the time, users will construct dfm objects from texts or a corpus, without calling tokens() as an intermediate step. Since tokens() is most likely to be used by more technical users, we have set its options to default to minimal intervention. This means that punctuation is tokenized as well, and that nothing is removed by default from the text being tokenized except inter-word spacing and equivalent characters.

Note that a tokens constructor also works on tokens objects, which allows setting additional options that will modify the original object. It is not possible, however, to change a setting to "un-remove" something that was removed from the input tokens object, however. For instance, tokens(tokens("Ha!", remove_punct = TRUE), remove_punct = FALSE) will not restore the "!" token. No warning is currently issued about this, so the user should use tokens.tokens() with caution.

Value

quanteda tokens class object, by default a serialized list of integers corresponding to a vector of types.

Dealing with URLs

URLs are tricky to tokenize, because they contain a number of symbols and punctuation characters. If you wish to remove these, as most people do, and your text contains URLs, then you should set what = "fasterword" and remove_url = TRUE. If you wish to keep the URLs, but do not want them mangled, then your options are more limited, since removing punctuation and symbols will also remove them from URLs. We are working on improving this behaviour.

See the examples below.

See Also

tokens_ngrams, tokens_skipgrams, as.list.tokens

Examples

txt1 <- c(doc1 = "This is a sample: of tokens.", doc2 = "Another sentence, to demonstrate how tokens works.")
tokens(txt1)
# removing punctuation marks and lowecasing texts
tokens(char_tolower(txt1), remove_punct = TRUE)
# keeping versus removing hyphens
tokens("quanteda data objects are auto-loading.", remove_punct = TRUE)
tokens("quanteda data objects are auto-loading.", remove_punct = TRUE, remove_hyphens = TRUE)
# keeping versus removing symbols
tokens("<tags> and other + symbols.", remove_symbols = FALSE)
tokens("<tags> and other + symbols.", remove_symbols = TRUE)
tokens("<tags> and other + symbols.", remove_symbols = FALSE, what = "fasterword")
tokens("<tags> and other + symbols.", remove_symbols = TRUE, what = "fasterword")
# examples with URLs - hardly perfect!
tokens(tokens(t2, remove_url = TRUE, remove_punct = TRUE))
tokens(tokens(t2, remove_url = FALSE, remove_punct = TRUE))
tokens(tokens(t2, remove_url = FALSE, remove_punct = TRUE, what = "fasterword"))
tokens(tokens(t2, remove_url = FALSE, remove_punct = FALSE, what = "fasterword"))

## MORE COMPARISONS

txt3 <- "#textanalysis is MY <3 4U @myhandle gr8 #stuff :-)")
tokens(txt3, remove_punct = TRUE)
tokens(txt3, remove_punct = TRUE, remove_twitter = TRUE)
tokens("great website http://textasdata.com", remove_url = FALSE)
tokens("great website http://textasdata.com", remove_url = TRUE)

## MORE COMPARISONS

txt4 <- c(text1="This is $10 in 999 different ways,\n up and down; left and right!",
 text2="@kenbenoit working: on #quanteda 2day\4ever, http://textasdata.com?page=123."))
tokens(txt4, verbose = TRUE)
tokens(txt4, remove_numbers = TRUE, remove_punct = TRUE)
tokens(txt4, remove_numbers = FALSE, remove_punct = TRUE)
tokens(txt4, remove_numbers = TRUE, remove_punct = FALSE)
tokens(txt4, remove_numbers = FALSE, remove_punct = FALSE)
tokens(txt4, remove_numbers = FALSE, remove_punct = FALSE, remove_separators = FALSE)
tokens(txt4, remove_numbers = TRUE, remove_punct = TRUE, remove_url = TRUE)

# character level

tokens("Great website: http://textasdata.com?page=123.", what = "character")

# sentence level

tokens(c("Kurt Vongeut said; only assholes use semi-colons.",
 "Today is Thursday in Canberra: It is yesterday in London.",
 "Today is Thursday in Canberra: \nIt is yesterday in London.",
 "To be? Or\nnot to be?"),
 what = "sentence")
tokens(data_corpus_inaugural[c(2,40)], what = "sentence")

# removing features (stopwords) from tokenized texts
txt5 <- char_tolower(c(mytext1 = "This is a short test sentence.",
 mytext2 = "Short.",
 mytext3 = "Short, shorter, and shortest."))
tokens(txt5, remove_puncut = TRUE)
tokens_remove(tokens(txt5, remove_punct = TRUE), stopwords("english"))

# ngram tokenization
tokens(txt5, remove_punct = TRUE, ngrams = 2)
tokens(txt5, remove_punct = TRUE, ngrams = 2, skip = 1, concatenator = " ")
tokens(txt5, remove_punct = TRUE, ngrams = 1:2)

# removing features from ngram token

tokens_remove(tokens(txt5, remove_punct = TRUE, ngrams = 1:2), stopwords("english"))
tokens_chunk  

Segment tokens object by chunks of a given size

Description

Segment tokens into new documents of equally sized token lengths, with the possibility of overlapping the chunks.

Usage

tokens_chunk(x, size, overlap = 0, use_docvars = TRUE)

Arguments

x  
tokens object whose token elements will be segmented into chunks

size  
integer; the token length of the chunks

overlap  
integer; the number of tokens in a chunk to be taken from the last overlap tokens from the preceding chunk

use_docvars  
if TRUE, repeat the docvar values for each chunk; if FALSE, drop the docvars in the chunked tokens

Value

A tokens object whose documents have been split into chunks of length size.

See Also

tokens_segment

Examples

txts <- c(doc1 = "Fellow citizens, I am again called upon by the voice of my country to execute the functions of its Chief Magistrate. ", doc2 = "When the occasion proper for it shall arrive, I shall endeavor to express the high sense I entertain of this distinguished honor."")
toks <- tokens(txts)
tokens_chunk(toks, size = 5)
tokens_chunk(toks, size = 5, overlap = 4)
**tokens_compound**

*Convert token sequences into compound tokens*

**Description**

Replace multi-token sequences with a multi-word, or "compound" token. The resulting compound tokens will represent a phrase or multi-word expression, concatenated with concatenator (by default, the "_" character) to form a single "token". This ensures that the sequences will be processed subsequently as single tokens, for instance in constructing a dfm.

**Usage**

```r
tokens_compound(
  x,
  pattern,
  concatenator = "_",
  valuetype = c("glob", "regex", "fixed"),
  case_insensitive = TRUE,
  join = TRUE
)
```

**Arguments**

- **x**: an input `tokens` object
- **pattern**: a character vector, list of character vectors, dictionary, or `collocations` object. See `pattern` for details.
- **concatenator**: the concatenation character that will connect the words making up the multi-word sequences. The default `_` is recommended since it will not be removed during normal cleaning and tokenization (while nearly all other punctuation characters, at least those in the Unicode punctuation class [P] will be removed).
- **valuetype**: the type of pattern matching: "glob" for "glob"-style wildcard expressions; "regex" for regular expressions; or "fixed" for exact matching. See `valuetype` for details.
- **case_insensitive**: logical; if TRUE, ignore case when matching. When `pattern` is a `collocations`, case-sensitive operation is significantly faster than case-insensitive operation.
- **join**: logical; if TRUE, join overlapping compounds into a single compound; otherwise, form these separately. See examples.

**Value**

A `tokens` object in which the token sequences matching `pattern` have been replaced by compound "tokens" joined by the concatenator.
Note

Patterns to be compounded (naturally) consist of multi-word sequences, and how these are expected in pattern is very specific. If the elements to be compounded are supplied as space-delimited elements of a character vector, wrap the vector in phrase. If the elements to be compounded are separate elements of a character vector, supply it as a list where each list element is the sequence of character elements.

See the examples below.

Examples

```r
txt <- "The United Kingdom is leaving the European Union."
toks <- tokens(txt, remove_punct = TRUE)

# character vector - not compounded
tokens_compound(toks, c("United", "Kingdom", "European", "Union"))

# elements separated by spaces - not compounded
tokens_compound(toks, c("United Kingdom", "European Union"))

# list of characters - is compounded
tokens_compound(toks, list(c("United", "Kingdom"), c("European", "Union")))

# elements separated by spaces, wrapped in phrase() - is compounded
tokens_compound(toks, phrase(c("United Kingdom", "European Union")))

# supplied as values in a dictionary (same as list) - is compounded
# (keys do not matter)
tokens_compound(toks, dictionary(list(key1 = "United Kingdom",
key2 = "European Union")))

# pattern as dictionaries with glob matches
tokens_compound(toks, dictionary(list(key1 = c("U* K*")), valuetype = "glob"))

# supplied as collocations - is compounded
colls <- tokens("The new European Union is not the old European Union.") %>%
textstat_collocations(size = 2, min_count = 1, tolower = FALSE)
tokens_compound(toks, colls, case_insensitive = FALSE)

# note the differences caused by join = FALSE
compounds <- list(c("the", "European"), c("European", "Union"))
tokens_compound(toks, pattern = compounds, join = TRUE)
tokens_compound(toks, pattern = compounds, join = FALSE)
```

Apply a dictionary to a tokens object

Convert tokens into equivalence classes defined by values of a dictionary object.
tokens_lookup

Usage

```
tokens_lookup(
  x,  
  dictionary, 
  levels = 1:5, 
  valuetype = c("glob", "regex", "fixed"), 
  case_insensitive = TRUE, 
  capkeys = !exclusive, 
  exclusive = TRUE, 
  nomatch = NULL, 
  nested_scope = c("key", "dictionary"), 
  verbose = quanteda_options("verbose")
)
```

Arguments

- **x**: tokens object to which dictionary or thesaurus will be supplied
- **dictionary**: the `dictionary`-class object that will be applied to `x`
- **levels**: integers specifying the levels of entries in a hierarchical dictionary that will be applied. The top level is 1, and subsequent levels describe lower nesting levels. Values may be combined, even if these levels are not contiguous, e.g. `levels = c(1:3)` will collapse the second level into the first, but record the third level (if present) collapsed below the first (see examples).
- **valuetype**: the type of pattern matching: "glob" for "glob"-style wildcard expressions; "regex" for regular expressions; or "fixed" for exact matching. See `valuetype` for details.
- **case_insensitive**: ignore the case of dictionary values if `TRUE` uppercase to distinguish them from other features
- **capkeys**: if `TRUE`, convert dictionary keys to uppercase to distinguish them from other features
- **exclusive**: if `TRUE`, remove all features not in dictionary, otherwise, replace values in dictionary with keys while leaving other features unaffected
- **nomatch**: an optional character naming a new key for tokens that do not matched to a dictionary values. If `NULL` (default), do not record unmatched tokens.
- **nested_scope**: how to treat matches from different dictionary keys that are nested. When one value is nested within another, such as "a b" being nested within "a b c", the `tokens_lookup()` will match the longer. When `nested_scope = "key"`, this longer-match priority is applied only within the key, while "dictionary" applies it across keys, matching only the key with the longer pattern, not the matches nested within that longer pattern from other keys. See Details.
- **verbose**: print status messages if `TRUE`

Details

Dictionary values may consist of sequences, and there are different methods of counting key matches based on values that are nested or that overlap.
When two different keys in a dictionary are nested matches of one another, the `nested_scope` options provide the choice of matching each key’s values independently (the "key") option, or just counting the longest match (the "dictionary" option). Values that are nested within the same key are always counted as a single match. See the last example below comparing the New York and New York Times for these two different behaviours.

Overlapping values, such as "a b" and "b a" are currently always considered as separate matches if they are in different keys, or as one match if the overlap is within the same key. Overlapped

See Also

tokens_replace

Examples

toks1 <- tokens(data_corpus_inaugural)
dict1 <- dictionary(list(country = "united states",
                      law=c("law\^\star", "constitution"),
                      freedom=c("free\^\star", "libert\^\star")))
dfm(tokens_lookup(toks1, dict1, valuetype = "glob", verbose = TRUE))
dfm(tokens_lookup(toks1, dict1, valuetype = "glob", verbose = TRUE, nomatch = "NONE"))

dict2 <- dictionary(list(country = "united states",
                      law = c("law", "constitution"),
                      freedom = c("freedom", "liberty")))
# dfm(applyDictionary(toks1, dict2, valuetype = "fixed"))
dfm(tokens_lookup(toks1, dict2, valuetype = "fixed"))

# hierarchical dictionary example
txt <- c(d1 = "The United States has the Atlantic Ocean and the Pacific Ocean.",
        d2 = "Britain and Ireland have the Irish Sea and the English Channel.")
toks2 <- tokens(txt)
dict3 <- dictionary(list(US = list(Countries = c("States"),
                               oceans = c("Atlantic", "Pacific")),
                         Europe = list(Countries = c("Britain", "Ireland"),
                                       oceans = list(west = "Irish Sea",
                                                     east = "English Channel"))))
tokens_lookup(toks2, dict3, levels = 1)
tokens_lookup(toks2, dict3, levels = 2)
tokens_lookup(toks2, dict3, levels = 1:2)
tokens_lookup(toks2, dict3, levels = 3)
tokens_lookup(toks2, dict3, levels = c(1,3))
tokens_lookup(toks2, dict3, levels = c(2,3))

# show unmatched tokens
tokens_lookup(toks2, dict3, nomatch = "_UNMATCHED")

# nested matching differences
dict4 <- dictionary(list(paper = "New York Times", city = "New York"))
toks4 <- tokens("The New York Times is a New York paper.")
tokens_lookup(toks4, dict4, nested_scope = "key", exclusive = FALSE)
tokens_lookup(toks4, dict4, nested_scope = "dictionary", exclusive = FALSE)
Create ngrams and skipgrams from tokens

Description

Create a set of ngrams (tokens in sequence) from already tokenized text objects, with an optional skip argument to form skipgrams. Both the ngram length and the skip lengths take vectors of arguments to form multiple lengths or skips in one pass. Implemented in C++ for efficiency.

Usage

```r
tokens_ngrams(x, n = 2L, skip = 0L, concatenator = "_")
char_ngrams(x, n = 2L, skip = 0L, concatenator = "_")
tokens_skipgrams(x, n, skip, concatenator = "_")
```

Arguments

- `x`: a tokens object, or a character vector, or a list of characters
- `n`: integer vector specifying the number of elements to be concatenated in each ngram. Each element of this vector will define a `n` in the `n`-gram(s) that are produced.
- `skip`: integer vector specifying the adjacency skip size for tokens forming the ngrams, default is 0 for only immediately neighbouring words. For skipgrams, `skip` can be a vector of integers, as the "classic" approach to forming skip-grams is to set `skip = k` where `k` is the distance for which `k` or fewer skips are used to construct the `n`-gram. Thus a "4-skip-n-gram" defined as `skip = 0:4` produces results that include 4 skips, 3 skips, 2 skips, 1 skip, and 0 skips (where 0 skips are typical n-grams formed from adjacent words). See Guthrie et al (2006).
- `concatenator`: character for combining words, default is `_` (underscore) character

Details

Normally, these functions will be called through `tokens(x, ngrams = ,...)`, but these functions are provided in case a user wants to perform lower-level ngram construction on tokenized texts.

`tokens_skipgrams` is a wrapper to `tokens_ngrams` that requires arguments to be supplied for both `n` and `skip`. For `k`-skip skipgrams, set `skip` to `0:k`, in order to conform to the definition of skip-grams found in Guthrie et al (2006): A `k` skip-gram is an ngram which is a superset of all ngrams and each `(k - i)` skipgram until `(k - i) == 0` (which includes 0 skip-grams).

Value

A tokens object consisting a list of character vectors of ngrams, one list element per text, or a character vector if called on a simple character vector
Note

char_ngrams is a convenience wrapper for a (non-list) vector of characters, so named to be consistent with quanteda's naming scheme.

Author(s)

Kohei Watanabe (C++) and Ken Benoit (R)

References


Examples

# ngrams
tokens_ngrams(tokens(c("a b c d e", "c d e f g")), n = 2:3)
toks <- tokens(c(text1 = "the quick brown fox jumped over the lazy dog"))
tokens_ngrams(toks, n = 1:3)
tokens_ngrams(toks, n = c(2,4), concatenator = " ")
tokens_ngrams(toks, n = c(2,4), skip = 1, concatenator = " ")
# on character
char_ngrams(letters[1:3], n = 1:3)

# skipgrams
toks <- tokens("insurgents killed in ongoing fighting")
tokens_skipgrams(toks, n = 2, skip = 0:1, concatenator = " ")
tokens_skipgrams(toks, n = 2, skip = 0:2, concatenator = " ")
tokens_skipgrams(toks, n = 3, skip = 0:2, concatenator = " ")

Description

Substitute token types based on vectorized one-to-one matching. Since this function is created for lemmatization or user-defined stemming. It support substitution of multi-word features by multi-word features, but substitution is fastest when pattern and replacement are character vectors and valuetype = "fixed" as the function only substitute types of tokens. Please use tokens_lookup with exclusive = FALSE to replace dictionary values.

Usage

tokens_replace(
  x, 
  pattern, 
  replacement,
tokens_replace

valuetype = "glob",
case_insensitive = TRUE,
verbose = quanteda_options("verbose")
)

Arguments

x 
tokens object whose token elements will be replaced

pattern 
a character vector or list of character vectors. See pattern for more details.

replacement 
a character vector or (if pattern is a list) list of character vectors of the same
length as pattern

valuetype 
the type of pattern matching: "glob" for "glob"-style wildcard expressions;
"regex" for regular expressions; or "fixed" for exact matching. See value-
type for details.

case_insensitive 
ignore case when matching, if TRUE

verbose 
print status messages if TRUE

See Also
tokens_lookup

Examples
toks1 <- tokens(data_corpus_irishbudget2010, remove_punct = TRUE)

# lemmatization
infle <- c("foci", "focus", "focused", "focuses", "focusing", "focussed", "focusses")
lemma <- rep("focus", length(infle))
toks2 <- tokens_replace(toks1, infle, lemma, valuetype = "fixed")
kwic(toks2, "focus*")

# stemming
type <- types(toks1)
stem <- char_wordstem(type, "porter")
toks3 <- tokens_replace(toks1, type, stem, valuetype = "fixed", case_insensitive = FALSE)
identical(toks3, tokens_wordstem(toks1, "porter"))

# multi-multi substitution
toks4 <- tokens_replace(toks1, phrase(c("Minister Deputy Lenihan")), phrase(c("Minister Deputy Conor Lenihan")))
kwic(toks4, phrase(c("Minister Deputy Conor Lenihan")))
Description

Sample tokenized documents randomly from a tokens object, with or without replacement. Works just as `sample` works, for document-level units (and their associated document-level variables).

Usage

```r
tokens_sample(x, size = ndoc(x), replace = FALSE, prob = NULL)
```

Arguments

- `x`: the `tokens` object whose documents will be sampled
- `size`: a positive number, the number of documents or features to select
- `replace`: logical; should sampling be with replacement?
- `prob`: a vector of probability weights for obtaining the elements of the vector being sampled.

Value

A `tokens` object with number of documents or features equal to `size`, drawn from the `tokens` `x`.

See Also

- `sample`

Examples

```r
set.seed(10)
toks <- tokens(data_corpus_inaugural[1:10])
head(toks)
head(tokens_sample(toks))
head(tokens_sample(toks, replace = TRUE))
```
tokens_select

Select or remove tokens from a tokens object

Description

These function select or discard tokens from a tokens objects. For convenience, the functions tokens_remove and tokens_keep are defined as shortcuts for tokens_select(x, pattern, selection = "remove") and tokens_select(x, pattern, selection = "keep"), respectively. The most common usage for tokens_remove will be to eliminate stop words from a text or text-based object, while the most common use of tokens_select will be to select tokens with only positive pattern matches from a list of regular expressions, including a dictionary.

Usage

tokens_select(
  x,
  pattern,
  selection = c("keep", "remove"),
  valuetype = c("glob", "regex", "fixed"),
  case_insensitive = TRUE,
  padding = FALSE,
  window = 0,
  min_nchar = NULL,
  max_nchar = NULL,
  verbose = quanteda_options("verbose")
)

tokens_remove(x, ...)

tokens_keep(x, ...)

Arguments

x tokens object whose token elements will be removed or kept
pattern a character vector, list of character vectors, dictionary, or collocations object. See pattern for details.
selection whether to "keep" or "remove" the tokens matching pattern
valuetype the type of pattern matching: "glob" for "glob"-style wildcard expressions; "regex" for regular expressions; or "fixed" for exact matching. See valuetyple for details.

Arguments

case_insensitive ignore case when matching, if TRUE
padding if TRUE, leave an empty string where the removed tokens previously existed. This is useful if a positional match is needed between the pre- and post-selected tokens, for instance if a window of adjacency needs to be computed.
window

integer of length 1 or 2; the size of the window of tokens adjacent to pattern that will be selected. The window is symmetric unless a vector of two elements is supplied, in which case the first element will be the token length of the window before pattern, and the second will be the token length of the window after pattern. The default is 0, meaning that only the pattern matched token(s) are selected, with no adjacent terms.

Terms from overlapping windows are never double-counted, but simply returned in the pattern match. This is because tokens_select never redefines the document units; for this, see kwic.

min_nchar, max_nchar

optional numerics specifying the minimum and maximum length in characters for tokens to be removed or kept; defaults are NULL for no limits. These are applied after (and hence, in addition to) any selection based on pattern matches.

verbose

if TRUE print messages about how many tokens were selected or removed

... additional arguments passed by tokens_remove and tokens_keep to tokens_select. Cannot include selection.

Value

a tokens object with tokens selected or removed based on their match to pattern

Examples

## tokens_select with simple examples
toks <- tokens(c("This is a sentence.", "This is a second sentence."), remove_punct = TRUE)
tokens_select(toks, c("is", "a", "this"), selection = "keep", padding = FALSE)
tokens_select(toks, c("is", "a", "this"), selection = "keep", padding = TRUE)
tokens_select(toks, c("is", "a", "this"), selection = "remove", padding = FALSE)
tokens_select(toks, c("is", "a", "this"), selection = "remove", padding = TRUE)

# how case_insensitive works
tokens_select(toks, c("is", "a", "this"), selection = "remove", case_insensitive = TRUE)
tokens_select(toks, c("is", "a", "this"), selection = "remove", case_insensitive = FALSE)

# use window
tokens_select(toks, "second", selection = "keep", window = 1)
tokens_select(toks, "second", selection = "remove", window = 1)
tokens_remove(toks, "is", window = c(0, 1))

# tokens_remove example: remove stopwords
txt <- c(wash1 <- "Fellow citizens, I am again called upon by the voice of my country to execute the functions of its Chief Magistrate.",
         wash2 <- "When the occasion proper for it shall arrive, I shall endeavor to express the high sense I entertain of this distinguished honor.")
tokens_remove(tokens(txt, remove_punct = TRUE), stopwords("english"))

# token_keep example: keep two-letter words
tokens_keep(tokens(txt, remove_punct = TRUE), "??")
tokens_split

Split tokens by a separator pattern

Description

Replaces tokens by multiple replacements consisting of elements split by a separator pattern, with
the option of retaining the separator. This function effectively reverses the operation of tokens_compound.

Usage

tokens_split(
  x,
  separator = " ",
  valuetype = c("fixed", "regex"),
  remove_separator = TRUE
)

Arguments

x a tokens object
separator a single-character pattern match by which tokens are separated
valuetype the type of pattern matching: "glob" for "glob"-style wildcard expressions;
"regex" for regular expressions; or "fixed" for exact matching. See value-
type for details.
remove_separator if TRUE, remove separator from new tokens

Examples

# undo tokens_compound()
toks1 <- tokens("pork barrel is an idiomatic multi-word expression")
tokens_compound(toks1, phrase("pork barrel"))
tokens_compound(toks1, phrase("pork barrel")) %>%
  tokens_split(separator = " ")

# similar to tokens(x, remove_hyphen = TRUE) but post-tokenization
toks2 <- tokens("UK-EU negotiation is not going anywhere as of 2018-12-24.")
tokens_split(toks2, separator = "-", remove_separator = FALSE)
tokens_subset

Extract a subset of a tokens

Description

Returns document subsets of a tokens that meet certain conditions, including direct logical operations on docvars (document-level variables). tokens_subset functions identically to subset.data.frame, using non-standard evaluation to evaluate conditions based on the docvars in the tokens.

Usage

tokens_subset(x, subset, select, ...)

Arguments

x tokens object to be subsetted
subset logical expression indicating the documents to keep: missing values are taken as false
select expression, indicating the docvars to select from the tokens; or a tokens object, in which case the returned tokens will contain the same documents in the same order as the original tokens, even if these are empty.
... not used

Value
tokens object, with a subset of documents (and docvars) selected according to arguments

See Also

subset.data.frame

Examples

corp <- corpus(c(d1 = "a b c d", d2 = "a a b e", d3 = "b b c e", d4 = "e e f a b"), docvars = data.frame(grp = c(1, 1, 2, 3)))
toks1 <- tokens(corp)
# selecting on a docvars condition
tokens_subset(toks1, grp > 1)
# selecting on a supplied vector
tokens_subset(toks1, c(TRUE, FALSE, TRUE, FALSE))

# selecting on a tokens
toks2 <- tokens(c(d1 = "a b b c", d2 = "b b c d"))
toks3 <- tokens(c(d1 = "x y z", d2 = "a b c c d", d3 = "x x x"))
tokens_subset(toks2, subset = toks3)
tokens_subset(toks2, subset = toks3[c(3,1,2)])
tokens_tolower

Convert the case of tokens

Description

tokens_tolower and tokens_toupper convert the features of a tokens object and re-index the types.

Usage

tokens_tolower(x, keep_acronyms = FALSE)
tokens_toupper(x)

Arguments

x the input object whose character/tokens/feature elements will be case-converted
keep_acronyms logical; if TRUE, do not lowercase any all-uppercase words (applies only to *_tolower functions)

Examples

# for a document-feature matrix
toks <- tokens(c(txt1 = "b A A", txt2 = "C C a b B"))
tokens_tolower(toks)
tokens_toupper(toks)

tokens_tortl

[Experimental] Change direction of words in tokens

Description

This function adds a Unicode direction mark to tokens types for punctuations and symbols to correct how right-to-left languages (e.g. Arabic, Hebrew, Persian, and Urdu) are printed in HTML-based consoles (e.g. R Studio). This is an experimental function subject to future change.

Usage

tokens_tortl(x)
char_tortl(x)

Arguments

x the input object whose punctuation marks will be modified by the direction mark
tokens_wordstem

Stem the terms in an object

Description

Apply a stemmer to words. This is a wrapper to `wordStem` designed to allow this function to be called without loading the entire `SnowballC` package. `wordStem` uses Martin Porter's stemming algorithm and the C `libstemmer` library generated by Snowball.

Usage

```r
tokens_wordstem(x, language = quanteda_options("language_stemmer"))
char_wordstem(x, language = quanteda_options("language_stemmer"))
dfm_wordstem(x, language = quanteda_options("language_stemmer"))
```

Arguments

- `x` a character, tokens, or dfm object whose word stems are to be removed. If tokenized texts, the tokenization must be word-based.
- `language` the name of a recognized language, as returned by `getStemLanguages`, or a two- or three-letter ISO-639 code corresponding to one of these languages (see references for the list of codes)

Value

- `tokens_wordstem` returns a `tokens` object whose word types have been stemmed.
- `char_wordstem` returns a `character` object whose word types have been stemmed.
- `dfm_wordstem` returns a `dfm` object whose word types (features) have been stemmed, and recombined to consolidate features made equivalent because of stemming.

References

- [http://snowball.tartarus.org/](http://snowball.tartarus.org/)
- [http://www.iso.org/iso/home/standards/language_codes.htm](http://www.iso.org/iso/home/standards/language_codes.htm) for the ISO-639 language codes

See Also

- `wordStem`
topfeatures

Examples

# example applied to tokens
txt <- c(one = "eating eater eaters eats ate",
        two = "taxing taxes taxed my tax return")
th <- tokens(txt)
tokens_wordstem(th)

# simple example
char_wordstem(c("win", "winning", "wins", "won", "winner"))

# example applied to a dfm
(origdfm <- dfm(txt))
dfm_wordstem(origdfm)

---

**topfeatures**

*Identify the most frequent features in a dfm*

Description

List the most (or least) frequently occurring features in a dfm, either as a whole or separated by document.

Usage

```
topfeatures(
  x,  
  n = 10,  
  decreasing = TRUE,  
  scheme = c("count", "docfreq"),  
  groups = NULL
)
```

Arguments

- **x**  
  the object whose features will be returned
- **n**  
  how many top features should be returned
- **decreasing**  
  If TRUE, return the n most frequent features; otherwise return the n least frequent features
- **scheme**  
  one of count for total feature frequency (within group if applicable), or docfreq for the document frequencies of features
- **groups**  
  either: a character vector containing the names of document variables to be used for grouping; or a factor or object that can be coerced into a factor equal in length or rows to the number of documents. See **groups** for details.
Value
A named numeric vector of feature counts, where the names are the feature labels, or a list of these if groups is given.

Examples
```
dfmat1 <- corpus_subset(data_corpus_inaugural, Year > 1980) %>%
  dfm(remove_punct = TRUE)
dfmat2 <- dfm_remove(dfmat1, stopwords("english"))

# most frequent features
topfeatures(dfmat1)
topfeatures(dfmat2)

# least frequent features
topfeatures(dfmat2, decreasing = FALSE)

# top features of individual documents
topfeatures(dfmat2, n = 5, groups = docnames(dfmat2))

# grouping by president last name
topfeatures(dfmat2, n = 5, groups = "President")

# features by document frequencies
tail(topfeatures(dfmat1, scheme = "docfreq", n = 200))
```

---

types

*Get word types from a tokens object*

Description
Get unique types of tokens from a tokens object.

Usage
```
types(x)
```

Arguments
- **x**: a tokens object

See Also
- featnames

Examples
```
toks <- tokens(data_corpus_inaugural)
types(toks)
```
Index

*Topic **bootstrap**
  bootstrap_dfm, 13

*Topic **character**
  corpus_segment, 22
corpus_trim, 25
tokens_tortl, 137

*Topic **collocations**
textstat_collocations, 99

*Topic **corpus**
corpus, 16
corpus_reshape, 20
corpus_sample, 21
corpus_segment, 22
corpus_subset, 24
corpus_trim, 25
docnames, 58
docvars, 59
head.corpus, 64
metacorpus, 67
metadoc, 68
texts, 98

*Topic **data**
data_char_sampletext, 26
data_char.ukimmig2010, 27
data_corpus_dailinoconf1991, 28
data_corpus_inaugural, 29
data_corpus_irishbudget2010, 30
data_dfm_lbgexample, 30
data_dictionary_LSD2015, 31

*Topic **dfm**
as.matrix.dfm, 9
bootstrap_dfm, 13
dfm, 33
dfm_lookup, 38
dfm_match, 40
dfm_sample, 42
dfm_select, 43
dfm_subset, 46
dfm_tfidf, 47
dfm_weight, 52
docfreq, 56
docnames, 58
featfreq, 63
head.dfm, 65

*Topic **experimental**
bootstrap_dfm, 13
textmodel_affinity, 78
textmodel_lsa, 80
tokens_tortl, 137

*Topic **plot**
textstat_frequency, 102

*Topic **textmodel**
textmodel_affinity, 78
textmodel_lsa, 80

*Topic **textplot**
textplot_influence, 88
textplot_keyness, 89
textplot_network, 90
textplot_scale1d, 92
textplot_wordcloud, 94
textplot_xray, 96

*Topic **textstat**
textstat_collocations, 99
textstat_keyness, 104
textstat_simil, 117

*Topic **tokens**
tokens, 120
tokens_chunk, 124
tokens_lookup, 126
tokens_sample, 132
tokens_split, 135
tokens_subset, 136
tokens_tortl, 137

*Topic **weighting**
dfm_tfidf, 47
docfreq, 56
featfreq, 63
+.tokens (as.tokents), 10
INDEX

DocumentTermMatrix, 7, 15

docvars, 16, 19, 25, 36, 37, 46, 59, 68, 102, 136
docvars<-(docvars), 59

fcm, 35, 36, 43, 44, 49, 60, 61, 62, 90, 91
fcm_compress(dfm_compress), 35
fcm_keep(dfm_select), 43
fcm_remove(dfm_select), 43
fcm_select, 91
fcm_sort, 62, 62
fcm_tolower(dfm_tolower), 49
fcm_toupper(dfm_tolower), 49
featfreq, 63
featnames, 36, 40, 51, 58, 63, 140
file, 54
frank, 102

getStemLanguages, 138
graph_from_adjacency_matrix, 91
groups, 34, 37, 93, 98, 102, 139

head.corpus, 64
head.dfm, 65

iconv, 54
influence.predict.textmodel_affinity, 88

is.collocations
  (textstat_collocations), 99
is.dfm(as.dfm), 7
is.dictionary, 55
is.dictionary(as.dictionary), 8
is.fcm(fcm), 60
is.kwic(kwic), 66
is.phrase(phrase), 74
is.tokens(as.tokens), 10

key-words-in-context, 6
keywords, 6
kwic, 16, 66, 96, 97, 134

lda.collapsed.gibbs.sampler, 15
lexical diversity measures, 6
list, 55

make.names, 118
Matrix, 7
matrix, 7

metacorpus, 16, 19, 67, 68
metacorpus<-(metacorpus), 67
metadoc, 19, 68
metadoc<-(metadoc), 68

ndoc, 19, 69
network, 91
nfeat, 61
nfeat(ndoc), 69
nscrabble, 70
nsentence, 71
nsyllable, 71
ntoken, 69, 73
ntype(ntoken), 73

options, 75

pattern, 22, 33, 41, 43, 66, 74, 125, 131, 133
pattern matches, 31
phrase, 67, 74, 74, 126
plot, 88
predict(), 87
predict.lm, 87
predict.textmodel_affinity, 78
predict.textmodel_lsa, 81
predict.textmodel_nb, 83
predict.textmodel_wordfish, 86
predict.textmodel_wordscores, 87, 88

quanteda(quanteda-package), 5
quanteda-package, 5, 29
quanteda_options, 18, 75
quantile, 50, 51

rbind.dfm, 35
readability indexes, 6

sample, 21, 42, 132
settings, 19
similarities, 6
SimpleCorpus, 16
size, 89
spacy_parse, 76
spacy_parse.corpus(spacyr-methods), 76
spacyr-methods, 76
sparsity, 77
stopwords, 33
stri_opts_brkiter, 71
stri_split_charclass, 121
stri_split_fixed, 121
stringi-search-boundaries, 121
subset, 18
subset.data.frame, 25, 46, 136
summary.corpus, 17
svds, 79, 80
tail.corpus (head.corpus), 64
tail.dfm (head.dfm), 65
TermDocumentMatrix, 7
textmodel_affinity, 78, 78, 88
textmodel_ca, 79, 92, 93
textmodel_isa, 80
textmodel_nb, 81
textmodel_wordfish, 84, 92, 93
textmodel_wordscores, 86, 88, 92, 93
textplot_influence, 88
textplot_keyness, 89
textplot_network, 90
textplot_scale1d, 92
textplot_wordcloud, 94
textplot_xray, 96
texts, 19, 98
texts<= (texts), 98
textstat_collocations, 99
textstat_dist, 9
textstat_dist (textstat_simil), 117
textstat_entropy, 101
textstat_frequency, 102
textstat_keyness, 89, 90, 104
textstat_lexdiv, 105, 111
textstat_readability, 109
textstat_simil, 9, 117
tf, 47
tokens, 10, 11, 20, 33, 34, 58–60, 66, 69, 71–73, 99, 100, 106, 120, 121, 122, 124, 125, 129, 131–138, 140
tokens_chunk, 124
tokens_compound, 125, 135
tokens_keep (tokens_select), 133
tokens_lookup, 11, 33, 39, 126, 130
tokens_ngrams, 121, 122, 129, 129
tokens_remove, 33, 100
tokens_remove (tokens_select), 133
tokens_replace, 130
tokens_sample, 132
tokens_segment, 124
tokens_select, 33, 133
tokens_skipgrams, 121, 122, 129
tokens_skipgrams (tokens_ngrams), 129
tokens_split, 135
tokens_subset, 136
tokens_tolower, 99, 137
tokens_tortl, 137
tokens_toupper (tokens_tolower), 137
tokens_wordstem, 75, 138
tolower, 14
topfeatures, 139
toupper, 14
types, 140
unlist, 11
unlist.tokens (as.tokens), 10
valuetype, 22, 34, 38, 44, 54, 66, 125, 127, 131, 133, 135
VCorpus, 16
wordStem, 138