Package ‘quanteda’

October 6, 2017

Version 0.99.12

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 (>= 3.4.0), methods

Imports utils, stats, Matrix (>= 1.2), data.table (>= 1.9.6),
    SnowballC, wordcloud, Rcpp (>= 0.12.12), RcppParallel,
    RSpectra, stringi, fastmatch, ggplot2 (>= 2.2.0), XML, yaml,
    lubridate, magrittr

LinkingTo Rcpp, RcppParallel, RcppArmadillo (>= 0.7.600.1.0)

Suggests knitr, rmarkdown, lda, proxy, topicmodels, tm (>= 0.6), slam,
    testthat, RColorBrewer, xtable, DT, ca, purrr

URL http://quanteda.io

Encoding UTF-8

BugReports https://github.com/kbenoit/quanteda/issues

LazyData TRUE

VignetteBuilder knitr

Collate 'RcppExports.R' 'View.R' 'bootstrap_dfm.R'
    'character-methods.R' 'convert.R' 'corpus-deprecated.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' 'dfm-classes.R' 'dfm-deprecated.R'
R topics documented:

'dfm-methods.R' 'dfm-print.R' 'dfm-subsetting.R' 'dfm.R'
'dfm_compress.R' 'dfm_group.R' 'dfm_lookup.R' 'dfm_sample.R'
'dfm_select.R' 'dfm_subset.R' 'dfm_trim.R' 'dfm_weight.R'
'dictionaries-deprecated.R' 'dictionaries-liwc_old.R'
'dictionaries.R' 'docnames.R' 'docvars.R' 'fcm-methods.R'
'fcm.R' 'joinTokens-deprecated.R' 'kwic.R' 'nfunctions.R'
'nscrabble.R' 'nsyllable.R' 'phrases.R' 'plots-deprecated.R'
'quanteda-documentation.R' 'quanteda_options.R'
'readtext-methods.R' 'regex2fixed.R' 'resample.R'
'selectFeatures-old.R' 'selectFeatures.R' 'settings.R'
'similarity.R' 'spacyr-methods.R' 'stopwords.R'
'textmodel-generics.R' 'textmodel-internal.R' 'textmodel_NB.R'
'textmodel_ca.R' 'textmodel_wordfish.R'
'textmodel_wordscores.R' 'textmodel_wordshoal.R'
'textplot_keyness.R' 'textplot_scale1d.R'
'textplot_wordcloud.R' 'textplot_xray.R'
'textstat-deprecated.R' 'textstat_collocations.R'
'textstat_dist.R' 'textstat_frequency.R' 'textstat_keyness.R'
'textstat_lexdiv.R' 'textstat_readability.R' 'textstat_simil.R'
'tokenize.R' 'tokenize_outtakes.R' 'tokens.R'
'tokens_compound.R' 'tokens_group.R' 'tokens_lookup.R'
'tokens_ngrams.R' 'tokens_old.R' 'tokens_segment.R'
'tokens_select.R' 'tolower-deprecated.R' 'tolower-functions.R'
'utils.R' 'wordstem.R' 'zzz.R'

RcppModules ngramMaker

RoxygenNote 6.0.1

SystemRequirements C++11

NeedsCompilation yes

Author Kenneth Benoit [aut, cre, cph],
Kohei Watanabe [ctb],
Paul Nulty [ctb],
Adam Obeng [ctb],
Haiyan Wang [ctb],
Benjamin Lauderdale [ctb],
Will Lowe [ctb]

Maintainer Kenneth Benoit <kbenoit@lse.ac.uk>

Repository CRAN

Date/Publication 2017-10-06 16:23:12 UTC
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>as.matrix.dfm</td>
<td>8</td>
</tr>
<tr>
<td>as.tokens</td>
<td>9</td>
</tr>
<tr>
<td>as.yaml</td>
<td>11</td>
</tr>
<tr>
<td>bootstrap_dfm</td>
<td>11</td>
</tr>
<tr>
<td>char_tolower</td>
<td>12</td>
</tr>
<tr>
<td>coef.textmodel</td>
<td>13</td>
</tr>
<tr>
<td>convert</td>
<td>14</td>
</tr>
<tr>
<td>corpus</td>
<td>15</td>
</tr>
<tr>
<td>corpus_reshape</td>
<td>18</td>
</tr>
<tr>
<td>corpus_sample</td>
<td>19</td>
</tr>
<tr>
<td>corpus_segment</td>
<td>20</td>
</tr>
<tr>
<td>corpus_subset</td>
<td>22</td>
</tr>
<tr>
<td>data_char_sampletext</td>
<td>23</td>
</tr>
<tr>
<td>data_char_ukimmig2010</td>
<td>24</td>
</tr>
<tr>
<td>data_corpus_inaugural</td>
<td>24</td>
</tr>
<tr>
<td>data_corpus_irishbudget2010</td>
<td>25</td>
</tr>
<tr>
<td>data_dfm_lbgexample</td>
<td>26</td>
</tr>
<tr>
<td>data_dictionary_LSD2015</td>
<td>27</td>
</tr>
<tr>
<td>dfm</td>
<td>28</td>
</tr>
<tr>
<td>dfm_compress</td>
<td>31</td>
</tr>
<tr>
<td>dfm_group</td>
<td>32</td>
</tr>
<tr>
<td>dfm_lookup</td>
<td>33</td>
</tr>
<tr>
<td>dfm_sample</td>
<td>34</td>
</tr>
<tr>
<td>dfm_select</td>
<td>35</td>
</tr>
<tr>
<td>dfm_sort</td>
<td>37</td>
</tr>
<tr>
<td>dfm_subset</td>
<td>38</td>
</tr>
<tr>
<td>dfm_tolower</td>
<td>39</td>
</tr>
<tr>
<td>dfm_trim</td>
<td>40</td>
</tr>
<tr>
<td>dfm_weight</td>
<td>42</td>
</tr>
<tr>
<td>dictionary</td>
<td>44</td>
</tr>
<tr>
<td>docnames</td>
<td>46</td>
</tr>
<tr>
<td>docvars</td>
<td>47</td>
</tr>
<tr>
<td>fcm</td>
<td>48</td>
</tr>
<tr>
<td>fcm_sort</td>
<td>50</td>
</tr>
<tr>
<td>featnames</td>
<td>51</td>
</tr>
<tr>
<td>head.corpus</td>
<td>51</td>
</tr>
<tr>
<td>head.dfm</td>
<td>52</td>
</tr>
<tr>
<td>is.dfm</td>
<td>53</td>
</tr>
<tr>
<td>kwic</td>
<td>54</td>
</tr>
<tr>
<td>metacorpus</td>
<td>55</td>
</tr>
<tr>
<td>metadoc</td>
<td>56</td>
</tr>
<tr>
<td>ndoc</td>
<td>57</td>
</tr>
<tr>
<td>nscrabble</td>
<td>58</td>
</tr>
<tr>
<td>nsentence</td>
<td>59</td>
</tr>
<tr>
<td>nsyllable</td>
<td>59</td>
</tr>
<tr>
<td>ntoken</td>
<td>60</td>
</tr>
<tr>
<td>phrase</td>
<td>61</td>
</tr>
<tr>
<td>quantedsa_options</td>
<td>62</td>
</tr>
</tbody>
</table>
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 \texttt{stringi} package, which is in turn built on C++ implementation of the ICU libraries for Unicode text handling, \texttt{quanteda} pays special attention to fast and correct implementation of Unicode and the handling of text in any character set.

\texttt{quanteda} is built for efficiency and speed, through its design around three infrastructures: the \texttt{stringi} package for text processing, the \texttt{data.table} package for indexing large documents efficiently, and the \texttt{Matrix} package for sparse matrix objects. If you can fit it into memory, \texttt{quanteda} will handle it quickly. (And eventually, we will make it possible to process objects even larger than available memory.)

\texttt{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 \texttt{quanteda} document-feature matrix ("dfm") can be easily analyzed using either \texttt{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 \texttt{quanteda}'s own functions for the "wordfish" and "Wordscores" models, direct use with the \texttt{ca} package for correspondence analysis, or scaling with the \texttt{austin} package) machine learning through a variety of other packages that take matrix or matrix-like inputs.

Additional features of \texttt{quanteda} include:

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

\textbf{Source code and additional information}

http://github.com/kbenoit/quanteda

\textbf{Author(s)}

\textbf{Maintainer}: Kenneth Benoit <kbenoit@lse.ac.uk> [copyright holder]

Other contributors:
See Also

Useful links:
- http://quanteda.io
- Report bugs at https://github.com/kbenoit/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.dictionary**

*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**

```r
as.dictionary(x)
```

```r
is.dictionary(x)
```
as.list.dist

Arguments

x       object to be coerced or checked; current legal values are a data.frame with the fields word and sentiment (as per the tidytext package)

Value

as.dictionary returns a 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

```r
## 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
afinn <- subset(sentiments, lexicon == "AFINN")
afinn["sentiment"] <-
  with(afinn,
    sentiment <- ifelse(score < 0, "negative",
                        ifelse(score > 0, "positive", "neutral"))
  )
with(afinn, table(score, sentiment))
as.dictionary(afinn)

## 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
```

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

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

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

Examples

```r
## Not run:
## compare to tm

# tm version
require(tm)
data("crude")
crude <- tm_map(crude, content_transformer(tolower))
crude <- tm_map(crude, remove_punctuation)
crude <- tm_map(crude, remove_numbers)
crude <- tm_map(crude, stemDocument)
tdm <- TermDocumentMatrix(crude)
findAssocs(tdm, c("oil", "opec", "xyz"), c(0.75, 0.82, 0.1))

# in quanteda
quantedaDfm <- as.dfm(t(as.matrix(tdm)))
as.list(textstat_simil(quantedaDfm, c("oil", "opec", "xyz"), margin = "features"), n = 14)

# in base R
corMat <- as.matrix(proxy::simil(as.matrix(quantedaDfm), by_rows = FALSE))
round(head(sort(corMat[, "oil"], decreasing = TRUE), 14), 2)
round(head(sort(corMat[, "opec"], decreasing = TRUE), 9), 2)

## End(Not run)
```

Description

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

Usage

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

## S3 method for class 'dfm'
as.data.frame(x, row.names = NULL, ...)
```
Arguments

x dfm to be coerced
...
row.names if FALSE, do not set the row names of the data.frame to the docnames of the dfm (default); or a vector of values to which the row names will be set.

Examples

# coercion to matrix
mydfm <- dfm(data_corpus_inaugural)
str(as.matrix(mydfm))

# coercion to a data.frame
inaugDfm <- dfm(data_corpus_inaugural[1:5])
as.data.frame(inaugDfm[, 1:10])
as.data.frame(inaugDfm[, 1:10], row.names = FALSE)

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, delimiter = "]")

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

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

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

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

is.tokens(x)

## S3 method for class 'tokens'
t1 + t2

## 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.

... for **c.tokens** only, **tokens** objects to be concatenated
- **recursive**: a required argument for **unlist** but inapplicable to **tokens** objects
- **use.names**: logical; preserve names if **true**. For **as.character** and **unlist** only.
- **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.
- **unlist** returns a simple vector 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.
- **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'))
lis <- list(c('The', 'United-States', 'has', 'the', 'Atlantic-Ocean',
             'and', 'the', 'Pacific-Ocean', '.'))
toks <- as.tokens(lis, 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

# combine tokens

c(toks1, toks2)
```
as.yaml

**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**

```r
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(), ".yaml")))
dictionary(file = yamlfile)

## End(Not run)
```

bootstrap_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, recombinining 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

# bootstrapping from the original text
txt <- c(textone = "This is a sentence. Another sentence. Yet another.",
        texttwo = "Premiere phrase. Deuxieme phrase.")
bootstrap_dfm(txt, n = 3)

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

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)
- ...: additional arguments passed to stringi functions, (e.g. stri_trans_tolower), such as locale
Examples

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

# with acronym preservation
txt2 <- 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)

Description

Extract text model coefficients for documents and features, in a manner similar to coef and coefficients. (coefficients is an alias for coef.)

Usage

coef.textmodel(object, ...)

Arguments

object a fitted or predicted text model object whose coefficients will be extracted
... unused

Value

Returns a list of named numeric vectors with the following elements:

coef_feature coefficients estimated for each feature
coef_feature_se standard errors estimated for each feature-level point estimate
coef_document coefficients estimated for each document
coef_document_se standard errors estimated for each document-level point estimate
coef_document_offset a document-level offset for applicable models
coef_feature_offset a feature-level offset for applicable models

An element that is not applicable for a particular object class will be NULL, for instance coef_documents has no meaning for a fitted wordscores object.
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. See also convert-wrappers for convenience functions for specific package converters.

Usage

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

Arguments

- `x`: 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
- `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.
- `...`: unused

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.

Note

There also exist a variety of converter shortcut commands, designed to mimic the idioms of the packages into whose format they convert. See convert-wrappers for details.
Examples

```r
mycorpus <- corpus_subset(data_corpus_inaugural, Year > 1970)
quantdfm <- dfm(mycorpus, verbose = FALSE)

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

# stm package format
stmdfm <- convert(quantdfm, to = "stm")
str(stmdfm)

# illustrate what happens with zero-length documents
quantdfm2 <- dfm(c(punctOnly = "!!!", mycorpus[-1]), verbose = FALSE)
rowSums(quantdfm2)
stmdfm2 <- convert(quantdfm2, to = "stm", docvars = docvars(mycorpus))
str(stmdfm2)

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

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

# lda package format
ladafm <- convert(quantdfm, to = "lda")
str(ladafm)

## 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, whose default document id is a variable identified by docid_field; the text of the document is a variable identified by textid_field; and other variables are imported as document-level meta-data. This matches the format of data.frames constructed by the readtext package.
- a kwic object constructed by kwic.
- a tm VWCorpus or SimpleCorpus class object, with the fixed metadata fields imported as docvars and corpus-level metadata imported as metacorpus information.
- a corpus object.
Usage

```r
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 = NULL, text_field = "text",
       metacorpus = NULL, compress = FALSE, ...)

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

## 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 cor-
pus 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 oper-
ations that require the texts to be extracted.
docid_field optional column index of a document identifier; if NULL, the constructor will use
the row.names of the data.frame (if found)
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 vari-
ables in the data.frame will be imported as docvars. This argument is only used
for data.frame objects (including those created by readtext).
**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.

```r
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).

**Author(s)**

Kenneth Benoit and Paul Nulty

**See Also**

`corpus-class`, `docvars`, `metadoc`, `metacorpus`, `settings`, `texts`, `ndoc`, `docnames`

**Examples**

```r
# 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
mydf <- 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))
mydf
summary(corpus(mydf, text_field = "some_text",
    metacorpus = list(source = "From a data.frame called mydf.")))
**corpus_sample**

### 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
corp <- 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(corp)
summary(corpus_reshape(corp, to = "sentences", showmeta = TRUE))

# example with inaugural corpus speeches
(corp2 <- corpus_subset(data_corpus_inaugural, Year>2004))
corp2_para <- corpus_reshape(corp2, to="paragraphs")
corp2_para
summary(corp2_para, 100, showmeta = TRUE)
## Note that Bush 2005 is recorded as a single paragraph because that text used a single
## \n to mark the end of a paragraph.
```

---

**corpus_sample** randomly sample documents from a corpus

### Description

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

### Usage

```r
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
- **replace**
  - Should sampling be with replacement?
- **prob**
  - A vector of probability weights for obtaining the elements of the vector being sampled.
- **by**
  - a grouping variable for sampling. Useful for resampling sub-document units such as sentences, for instance by specifying by = "document"
- **...**
  - unused
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

```r
# sampling from a corpus
summary(corpus_sample(data_corpus_inaugural, 5))
summary(corpus_sample(data_corpus_inaugural, 10, replace = TRUE))

# sampling sentences within document
doccorpus <- corpus(c(one = "Sentence one. Sentence two. Third sentence.",
                      two = "First sentence, doc2. Second sentence, doc2."))
sentcorpus <- corpus_reshape(doccorpus, to = "sentences")
texts(sentcorpus)
texts(corpus_sample(sentcorpus, 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 (a "tag").

Usage

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

char_segment(x, pattern = "###", valuetype = c("glob", "regex", "fixed"),
             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, collocations, or dfm. 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.
- **extract_pattern** extracts matched patterns from the texts and save in docvars if TRUE
**corpus_segment**

- **pattern_position**
  - either "before" or "after", depending on whether the pattern precedes the text (as with a tag) or follows the text (as with punctuation delimiters)
- **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
corp <- corpus(c("##INTRO This is the introduction.
##DOC1 This is the first document. Second sentence in Doc 1.
##DOC3 Third document starts here. End of third document.",
"##INTRO Document ##NUMBER Two starts before ##NUMBER Three.")
corp_seg <- corpus_segment(corp, "##")
cbind(texts(corp_seg), docvars(corp_seg), metadoc(corp_seg))

# segmenting a transcript based on speaker identifiers
corp2 <- corpus("Mr. Smith: Text.\nMrs. Jones: More text.\nMr. Smith: I’m speaking, again."")
corp_seg2 <- corpus_segment(corp2, pattern = "\b[A-Z]+\s[A-Z][a-z]+:\", valuetype = "regex")
cbind(texts(corp_seg2), docvars(corp_seg2), metadoc(corp_seg2))

# segmenting a corpus using crude end-of-sentence segmentation
corp_seg3 <- corpus_segment(corp, pattern = ".", valuetype = "fixed", pattern_position = "after", extract_pattern = FALSE)
cbind(texts(corp_seg3), docvars(corp_seg3), metadoc(corp_seg3))

## 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\\{-\s{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)
data_char_sampletext

Description

This is a long paragraph (2,914 characters) of text taken from a debate on the Irish budget in Dáil Éireann by Socialist Teachta Dála (TD) Joe Higgins, delivered December 8, 2011.

Usage

data_char_sampletext

Format

character 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_inaugural  US presidential inaugural address texts

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.

Source

data_dfm_lbgexample

References


Examples

summary(data_corpus_irishbudgetRP1PI)

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

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

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 preceeded by a negation (used to convey negative sentiment)
negative 2,860 word patterns indicating a negative word preceeded 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

The objectives, development and reliability of the dictionary are discussed in detail in Young and Soroka (2012). Please cite this article when using the Lexicoder Sentiment Dictionary and related resources. Young, Lori and Stuart Soroka. 2012. *Lexicoder Sentiment Dictionary*. Available at http://lexicoder.com.


Examples

```r
# simple example
taxt <- "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" "."

# on larger examples - notice that few negations are used
dfm(data_char_ukimmig2010, dictionary = data_dictionary_LSD2015)
kwic(data_char_ukimmig2010, "not")
```

dfm  
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 tokenize. Note: select = "^@[a-zA-Z0-9]+\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 value-type 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 behavior 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.

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
corpus_postinaug <- corpus_subset(data_corpus_inaugural, Year > 1980)
dfm(corpus_postinaug)
dfm(corpus_postinaug, tolower = FALSE)

# grouping documents by docvars in a corpus
```r

dfm(corpus_post01inaug, groups = "President", verbose = TRUE)

# with English stopwords and stemming
dfm(corpus_post01inaug, remove = stopwords("english"), stem = TRUE, verbose = TRUE)
# works for both words in ngrams too
dfm("Banking industry", stem = TRUE, ngrams = 2, verbose = FALSE)

# with dictionaries
corpus_post1900inaug <- corpus_subdata(data_corpus_1900inaug, Year > 1900)
mydict <- dictionary(list(christmas = c("Christmas", "Santa", "holiday"),
opposition = c("Opposition", "reject", "notincorporus"),
taxing = "taxing",
taxation = "taxation",
taxregex = "tax",
country = "states"))
dfm(corpus_post1900inaug, dictionary = mydict)

# removing stopwords
testText <- "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."
testCorpus <- corpus(testText)
# note: "also" is not in the default stopwords("english")
featnames(dfm(testCorpus, select = stopwords("english")))
# for ngrams
featnames(dfm(testCorpus, ngrams = 2, select = stopwords("english"), remove_punct = TRUE))
featnames(dfm(testCorpus, ngrams = 1:2, select = stopwords("english"), remove_punct = TRUE))

# removing stopwords before constructing ngrams
tokensAll <- tokens(char_tolower(testText), remove_punct = TRUE)
tokensNoStopwords <- removeFeatures(tokensAll, stopwords("english"))
tokensNgramsNoStopwords <- tokens_ngrams(tokensNoStopwords, 2)
featnames(dfm(tokensNgramsNoStopwords, verbose = FALSE))

# keep only certain words
dfm(testCorpus, select = ".*s", verbose = FALSE)  # keep only words ending in "s"
dfm(testCorpus, select = "s\$", valuetype = "regex", verbose = FALSE)

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

# for a dfm
dfm1 <- dfm(data_corpus_irishbudget2010)
dfm2 <- dfm(dfm1,
groups = ifelse(docvars(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.df` or `rbind.df` 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"))
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 recombed 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 recombed 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
mat <- rbind(dfm(c("b A A", "C C a b B"), tolower = FALSE), dfm("A C C C C", tolower = FALSE))
colnames(mat) <- char_tolower(featnames(mat))
mat
dfm_compress(mat, margin = "documents")
dfm_compress(mat, margin = "features")
dfm_compress(mat)
```
# no effect if no compression needed
compactdfm <- dfm(data_corpus_inaugural[1:5])
dim(compactdfm)
dim(dfm_compress(compactdfm))

# compress an fcm
myfcm <- 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(myfcm)

txt <- c("The fox JUMPED over the dog.",
         "The dog jumped over the fox.")
toks <- tokens(txt, remove_punct = TRUE)
myfcm <- fcm(toks, context = "document")
colnames(myfcm) <- rownames(myfcm) <- tolower(colnames(myfcm))
colnames(myfcm)[5] <- rownames(myfcm)[5] <- "fox"
myfcm
fcm_compress(myfcm)

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)

Arguments

x

da 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.

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. This currently erases any docvars
in the dfm.
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

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

# equivalent
dfm(mydfm, groups = "grp")
dfm(mydfm, 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.
- **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.

Examples

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

# glob format
dfm_lookup(myDfm, myDict, valuetype = "glob")
dfm_lookup(myDfm, myDict, valuetype = "glob", case_insensitive = FALSE)

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

# fixed format: no pattern matching
dfm_lookup(myDfm, myDict, valuetype = "fixed")

# show unmatched tokens
dfm_lookup(myDfm, myDict, nomatch = "_UNMATCHED")
```
Usage

```r
dfm_sample(x, size = ndoc(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.
- `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

```r
set.seed(10)
myDfm <- dfm(data_corpus_inaugural[1:10])
head(myDfm)
head(dfm_sample(myDfm))
head(dfm_sample(myDfm, replace = TRUE))
head(dfm_sample(myDfm, margin = "features"))
```

---

**dfm_select**

*select features from a dfm or fcm*

**Description**

This function selects or discards 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, selection = c("keep", "remove"), valuetype = c("glob", "regex", "fixed"), case_insensitive = TRUE, min_nchar = 1L, max_nchar = 63L, verbose = quanteda_options("verbose"), ...)
```
dfm_remove(x, pattern, ...)

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

fcm_remove(x, pattern, ...)

**Arguments**

- **x**: the dfm or fcm object whose features will be selected
- **pattern**: a character vector, list of character vectors, dictionary, collocations, or dfm. 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.
- **case_insensitive**: ignore the case of dictionary values if TRUE
- **min_nchar, max_nchar**: numerics specifying the minimum and maximum length in characters for features to be removed or kept; defaults are 1 and 79. (Set `max_nchar` to NULL for no upper limit.) 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 *_remove to *_select functions

**Details**

dfm_remove and fcm_remove are simply a convenience wrappers to calling dfm_select and fcm_select with `selection = "remove"`.

**Value**

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

When `pattern` is a dfm object, then the returned object will be identical in its feature set to the dfm supplied as the pattern argument. This means that any features in `x` not in the dfm provided as pattern will be discarded, and that any features in found in the dfm supplied as pattern but not found in `x` will be added with all zero counts. Because selecting on a dfm is designed to produce a selected dfm with an exact feature match, when `pattern` is a dfm object, then the following settings are always used: `case_insensitive` = FALSE, and `valuetype` = "fixed".

Selecting on a dfm 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. See examples.

**Note**

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

```r
defm <- dfm(c("My Christmas was ruined by your opposition tax plan.",
  "Does the United_States or Sweden have more progressive taxation?"),
  tolower = FALSE, verbose = FALSE)
defm <- dfm(c("United_States", "Sweden", "France"),
  wordsEndingInY = c("by", "my"),
  notintext = "blahblah")
dfm_select(dfm, dictionary(list(countries = c("United_States", "Sweden", "France"),
  wordsEndingInY = c("by", "my"),
  notintext = "blahblah"))
dfm_select(dfm, dictionary(countries = c("United_States", "Sweden", "France"),
  wordsEndingInY = c("by", "my"),
  notintext = "blahblah"))
# select based on character length
dfm_select(dfm, min_nchar = 5)

# selecting on a dfm
txts <- c("This is text one", "The second text", "This is text three")
(dfm1 <- dfm(txts[1:2]))
(dfm2 <- dfm(txts[2:3]))
(dfm3 <- dfm_select(dfm1, dfm2, valuetype = "fixed", verbose = TRUE))
setequal(Featnames(dfm1), Featnames(dfm2))

# select based on character length
dfm_select(dfm, min_nchar = 5)

tmpdfm <- dfm(c("This is a document with lots of stopwords.",
  "No if, and, or but about it: lots of stopwords.",
  verbose = FALSE)

tmpdfm

dfm_remove(tmpdfm, stopwords("english"))
toks <- tokens(c("this contains lots of stopwords",
  "no if, and, or but about it: lots"),
  remove_punct = TRUE)
tmpfcm <- fcm(toks)
tmpfcm

cfm_remove(tmpfcm, stopwords("english"))
```

---

**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**

```r
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

```r
dtm <- dfm(data_corpus_inaugural)
head(dtm)
head(dfm_sort(dtm))
head(dfm_sort(dtm, 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

```r
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, in which case the returned `dfm` will contain the same documents as the original dfm, even if these are empty. See Details.
- **...**: not used
dfm_tolower

Details

To select or subset features, see dfm_select instead.

When select is a dfm, then the returned dfm will be equal in row dimensions 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

testcorp <- 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)))
testdfm <- dfm(testcorp)
# selecting on a docvars condition
dfm_subset(testdfm, grp > 1)
# selecting on a supplied vector
dfm_subset(testdfm, 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_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)

... additional arguments passed to stringi functions, (e.g. stri_trans_tolower), such as locale

Details

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

Examples

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

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

dfm_trim is 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 frequencies, but may also be in terms of maximum frequencies.
Setting a combination of minimum and maximum frequencies will select features based on a range.

Usage

dfm_trim(x, min_count = 1, min_docfreq = 1, max_count = NULL,
         max_docfreq = NULL, sparsity = NULL,
         verbose = quanteda_options("verbose"))
Arguments

- **x**
  - A dfm object
- **min_count, max_count**
  - minimum/maximum count or fraction of features across all documents, below/above which features will be removed
- **min_docfreq, max_docfreq**
  - minimum/maximum number or fraction of documents in which a feature appears, below/above which features will be removed
- **sparsity**
  - Equivalent to 1 - min_docfreq, included for comparison with tm
- **verbose**
  - Print messages

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.

Author(s)

Ken Benoit and Paul Nulty, with some inspiration from Will Lowe (see trim from the austin package)

See Also

dfm_select, dfm_sample

Examples

```
(myDfm <- dfm(data_corpus_inaugural[1:5]))

# keep only words occurring \( \geq 10 \) times and in \( \geq 2 \) docs
dfm_trim(myDfm, min_count = 10, min_docfreq = 2)

# keep only words occurring \( \geq 10 \) times and in at least 0.4 of the documents
dfm_trim(myDfm, min_count = 10, min_docfreq = 0.4)

# keep only words occurring \( \leq 10 \) times and in \( \leq 2 \) docs
dfm_trim(myDfm, max_count = 10, max_docfreq = 2)

# keep only words occurring \( \leq 10 \) times and in at most 3/4 of the documents
dfm_trim(myDfm, max_count = 10, max_docfreq = 0.75)

# keep only words occurring at least 0.01 times and in \( \geq 2 \) documents
dfm_trim(myDfm, min_count = .01, min_docfreq = 2)
```
# keep only words occurring 5 times in 1000, and in 2 of 5 of documents
dfm_trim(myDfm, min_docfreq = 0.4, min_count = 0.005)

## Not run:
# compare to removeSparseTerms from the tm package
if (require(tm)) {
  (tmdtm <- convert(myDfm, "tm"))
  removeSparseTerms(tmdtm, 0.7)
  dfm_trim(td, min_docfreq = 0.3)
  dfm_trim(td, sparsity = 0.7)
}

## End(Not run)

dfm_weight

## weight the feature frequencies in a dfm

### Description

Returns a document by feature matrix with the feature frequencies weighted according to one of several common methods. Some shortcut functions that offer finer-grained control are:

- **tf** compute term frequency weights
- **tfidf** compute term frequency-inverse document frequency weights
- **docfreq** compute document frequencies of features

### Usage

```r
dfm_weight(x, type = c("frequency", "relfreq", "relmaxfreq", "logfreq", "tfidf"), weights = NULL)
```

```r
dfm_smooth(x, smoothing = 1)
```

### Arguments

- **x** document-feature matrix created by `dfm`
- **type** a label of the weight type:
  - "frequency" integer feature count (default when a dfm is created)
  - "relfreq" the proportion of the feature counts of total feature counts (aka relative frequency)
  - "relmaxfreq" the proportion of the feature counts of the highest feature count in a document
  - "logfreq" take the logarithm of 1 + the feature count, for base 10
weights if type 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).

smoothing constant added to the dfm cells for smoothing, default is 1

Value
dfm_weight returns the dfm with weighted values.
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.

Note
For finer grained control, consider calling the convenience functions directly.

Author(s)
Paul Nulty and Kenneth Benoit

References

See Also
tf, tfidf, docfreq

Examples
dtm <- dfm(data_corpus_inaugural)

x <- apply(dtm, 1, function(tf) tf/max(tf))
topfeatures(dtm)
normDtm <- dfm_weight(dtm, "relfreq")
topfeatures(normDtm)
maxTfDtm <- dfm_weight(dtm, type = "relmaxfreq")
topfeatures(maxTfDtm)
logTfDtm <- dfm_weight(dtm, type = "logfreq")
topfeatures(logTfDtm)
tfidfDtm <- dfm_weight(dtm, type = "tfidf")
topfeatures(tfidfDtm)

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

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

# smooth the dfm
dfm_smooth(mydfm, 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:

1. "wordstat" format used by Provalis Research’s Wordstat software

2. "LIWC" format used by the Linguistic Inquiry and Word Count software

3. "yoshikoder" format used by Yoshikoder software

4. "lexicoder" format used by Lexicoder

5. "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 \[\text{and} \] and \[\text{c} \]. operating the same as the equivalent list operators. Dictionaries can be coerced from lists using \texttt{as.dictionary}, coerced to named lists of characters using \texttt{as.list}, and checked using \texttt{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

\texttt{dfm, as.dictionary, as.list, is.dictionary}

Examples

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

# subset a dictionary
mydict[1:2]
mydict[["christmas", "opposition"]]
mydict[["opposition"]]

# combine dictionaries
c(mydict["christmas"], mydict["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()))
lgdict <- dictionary(file = paste(td, "LaverGarry.cat", sep = "/"))
head(dfm(data_corpus_inaugural, dictionary = lgdict))

# import a LIWC formatted dictionary from http://www.moralfoundations.org
```
docnames

Description

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

Usage

docnames(x)

docnames(x) <- 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.

See Also

featnames

Examples

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

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

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

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

**Index access to docvars in a corpus**

Another way to access and set docvars is through indexing of the corpus `j` element, such as `data_corpus_irishbudget2010[, c("foren","name")]`; or, for a single docvar, `data_corpus_irishbudget2010[["name"]]

The latter also permits assignment, including the easy creation of new document variables, e.g. `data_corpus_irishbudget2010[["newvar"]] <-1:ndoc(data_corpus_irishbudget2010). See [.corpus for details.

**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

# 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))

# alternative using indexing
head(corp[,, "Year"])
corp["President2"] <- paste("prezTwo", 1:ndoc(corp), sep = "")
head(docvars(corp))

fcm 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 = 1L, ordered = FALSE, span_sentence = TRUE, 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 length(weights)

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".

span_sentence if FALSE, then word windows will not span sentences

tri if TRUE return only upper triangle (including diagonal)

Details

The function \texttt{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 \textit{document}, \textit{sentences} within documents, \textit{syntactic relationships} between features (nouns within a sentence, for instance), or according to a \textit{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).

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

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

\texttt{fcm} also provides "boolean" counting within the context of "window", which differs from the counting within "document".

\texttt{is.fcm(x)} returns TRUE if and only if its x is an object of type \texttt{fcm}.

Author(s)

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

References


Examples

\begin{verbatim}
# see http://bit.ly/29b2z0A
txt <- "A D A C E A D F E B A C E D"
fcm(txt, context = "window", window = 2)
fcm(txt, context = "window", count = "weighted", window = 3)
fcm(txt, context = "window", count = "weighted", window = 3,
\end{verbatim}
weights = c(3, 2, 1), ordered = TRUE, tri = FALSE)

# with multiple documents
txts <- c("a a b b c", "a c e", "a c e f g")
fcm(txts, context = "document", count = "frequency")
fcm(txts, context = "document", count = "boolean")
fcm(txts, context = "window", window = 2)

# from tokens
txt <- c("The quick brown fox jumped over the lazy dog.",
         "The dog jumped and ate the fox.")
toks <- tokens(char_tolower(txt), 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)

Ken Benoit

Examples

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

# with tri = TRUE
myfcm <- fcm(tokens(c("A X Y C B A", "X Y C A B B")), tri = TRUE)
### 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

```r
featnames(x)
```

#### Arguments

- `x`  
  the dfm whose features will be extracted

#### Value

Character vector of the feature labels

#### Examples

```r
inaugDfm <- dfm(data_corpus_inaugural, verbose = FALSE)

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

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

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

### head.corpus

*return the first or last part of a corpus*

#### Description

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

```r
head.corpus(..., n = NULL, from = NULL, to = NULL)
```
Usage

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

## 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

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

**head.df m**

*return the first or last part of a dfm*

Description

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

Usage

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

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

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.
is.dfm

nfeature the number of features to return, where the resulting object will contain the first ncol features

... additional arguments passed to other functions

Value

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

Examples

head(data_dfm_lbgexample, 3, nfeature = 5)
head(data_dfm_lbgexample, -4)

tail(data_dfm_lbgexample)
tail(data_dfm_lbgexample, n = 3, nfeature = 4)

---

is.dfm coercion and checking functions for dfm objects

Description

Check for a dfm, or convert a matrix into a dfm.

Usage

is.dfm(x)
as.dfm(x)

Arguments

x a dfm object

Value

is.dfm returns TRUE if and only if its argument is a dfm.
as.dfm coerces a matrix or data.frame to a dfm. Row names are used for docnames, and column names for featnames, of the resulting dfm.

See Also

as.data.frame.dfm, as.matrix.dfm
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

```r
kwic(x, pattern, window = 5, valuetype = c("glob", "regex", "fixed"),
  case_insensitive = TRUE, join = FALSE, ...)

is.kwic(x)
```

## S3 method for class 'kwic'

```r
as.tokens(x, ...)
```

Arguments

- `x`: a character, corpus, or tokens object
- `pattern`: a character vector, list of character vectors, dictionary, collocations, or dfm. 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.
- `case_insensitive`: match without respect to case if TRUE
- `join`: join adjacent keywords in the concordance view if TRUE
- `...`: additional arguments passed to tokens, for applicable object types

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`.

`as.tokens.kwic` converts the kwic object into a tokens object, with each new "document" consisting of one keyword match, and the contents of the `pre`, `keyword`, and `post` fields forming the tokens. This is one way to save the output for subsequent usage; another way is to form a corpus from the return object.
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.

Author(s)

Kenneth Benoit and Kohei Watanabe

Examples

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

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

mykwic <- kwic(data_corpus_inaugural, "provident\*")
is.kwic(mykwic)
is.kwic("Not a kwic")

---

metacorpus get or set corpus metadata

Description

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

Usage

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

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

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

    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

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

Description

Get the number of documents or features in an object.

Usage

ndoc(x)
nfeature(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 a corpus, dfm, or tokens object, or a readtext object from the readtext package
nfeature returns the number of features in a dfm

nfeature 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
nfeature(dfm(corpus_subset(data_corpus_inaugural, Year > 1980), remove_punct = FALSE))
nfeature(dfm(corpus_subset(data_corpus_inaugural, Year > 1980), remove_punct = TRUE))

---

nscramble  

\*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

nscramble(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 Scabble 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

nscramble(c("muzjiks", "excellency"))
nscramble(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**

```r
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**

```r
# 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

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

Arguments

x character vector or tokens object whose syllables will be counted
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().

Examples

# 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

\textit{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

Count 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

```r
# 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
ntoken(corpus_subset(data_corpus_inaugural, Year<1806), remove_punct = TRUE)
ntype(corpus_subset(data_corpus_inaugural, Year<1806), remove_punct = TRUE)
ntoken(dfm(corpus_subset(data_corpus_inaugural, Year<1800)))
ntype(dfm(corpus_subset(data_corpus_inaugural, Year<1800)))
```

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**

```r
phrase(x)
```

```r
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**

```r
# 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**

```r
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 use this as the setting in all functions that use parallelization
- `print_dfm_max_ndoc` integer; specifies the number of documents to display when using the defaults for printing a dfm
- `print_dfm_max_nfeature` 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_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_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

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("verbose" = 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 of methods defined in the quanteda package

Description
Extensions to quanteda functions. You must have attached **quanteda** for these to work.

Arguments

- `x`: an object returned by `spacy_parse`
- `...`: unused

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

Examples

```r
## Not run:
require(spacyr)
spacy_initialize()
txt <- c(doc1 = "And now, now, now for something completely different.",
         doc2 = "Jack and Jill are children.")
parsed <- spacy_parse(txt)
ntype(parsed)
ntoken(parsed)
ndoc(parsed)
docnames(parsed)

## End(Not run)
```

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

`sparsity(x)`
stopwords

Arguments

x the document-feature matrix

Examples

inaug_dfm <- dfm(data_corpus_inaugural, verbose = FALSE)
sparsity(inaug_dfm)
sparsity(dfm_trim(inaug_dfm, min_count = 5))

Description

This function retrieves stopwords from the type specified in the kind argument and returns the
stopword list as a character vector. The default is English.

Usage

stopwords(kind = quanteda_options("language_stopwords"))

Arguments

kind The pre-set kind of stopwords (as a character string). Allowed values are english,
SMART, danish, french, greek, hungarian, norwegian, russian, swedish,
catalan, dutch, finnish, german, italian, portuguese, spanish, arabic.

Details

The stopword list is an internal data object named data_char_stopwords, which consists of En-
glish stopwords from the SMART information retrieval system (obtained from Lewis et. al. (2004)
and a set of stopword lists from the Snowball stemmer project in different languages (see http:
//snowballstem.org/projects.html). See data_char_stopwords for details.

Value

a character vector of stopwords

A note of caution

Stop words are an arbitrary choice imposed by the user, and accessing a pre-defined list of words to
ignore does not mean that it will perfectly fit your needs. You are strongly encouraged to inspect the
list and to make sure it fits your particular requirements.
Source


Additional stopword lists are taken from the Snowball stemmer project in different languages (see http://snowballstem.org/projects.html).

The Greek stopwords were supplied by Carsten Schwemmer (see GitHub issue #282).

Examples

head(stopwords("english"))
head(stopwords("italian"))
head(stopwords("arabic"))
head(stopwords("SMART"))

# adding to the built-in stopword list
toks <- tokens("The judge will sentence Mr. Adams to nine years in prison", remove_punct = TRUE)
tokens_remove(toks, c(stopwords("english"), "will", "mr", "nine"))

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, and returns a special class of ca object.

Usage

textmodel_ca(x, smooth = 0, nd = NA, sparse = FALSE, threads = 1, 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
threads the number of threads to be used; set to 1 to use a serial version of the function; only applicable when sparse = TRUE
residual_floor specifies the threshold for the residual matrix for calculating the truncated svd. Larger value will reduce memory and time cost but might sacrifice the accuracy; only applicable when sparse = TRUE
Details

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

Note

Setting threads larger than 1 (when sparse = TRUE) will trigger parallel computation, which retains sparsity of all involved matrices. You may need to 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


Examples

```r
ieDfm <- dfm(data_corpus_irishbudget2010)
wca <- textmodel_ca(ieDfm)
summary(wca)
```

Description

Naïve Bayes classifier for texts

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

Usage

```r
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 distribution on texts; see Details
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 tf(x, "boolean").
... more arguments passed through

Value
A list of return values, consisting of:
call original function call
PwGc probability of the word given the class (empirical likelihood)
Pc class prior probability
PcGw posterior class probability given the word
Pw baseline probability of the word
data list consisting of x training class, and y test class
distribution the distribution argument
prior argument passed as a prior
smooth smoothing parameter

Predict Methods
A predict method is also available for a fitted Naive Bayes object, see predict.textmodel_NB_fitted.

Author(s)
Kenneth Benoit

References
Jurafsky, Daniel and James H. Martin. (2016) Speech and Language Processing. Draft of November

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

## replicate IIR p261 prediction for test set (document 5)
(nb.p261 <- textmodel_NB(trainingset, trainingclass, prior = "docfreq"))
predict(nb.p261, newdata = trainingset[5, ])

# contrast with other priors
predict(textmodel_NB(trainingset, trainingclass, prior = "uniform"))
predict(textmodel_NB(trainingset, trainingclass, prior = "termfreq"))

## replicate IIR p264 Bernoulli Naive Bayes
(nb.p261.bern <- textmodel_NB(trainingset, trainingclass, distribution = "Bernoulli",
                               prior = "docfreq"))
predict(nb.p261.bern, newdata = trainingset[5, ])

---

textmodel_wordfish wordfish text model

**Description**


**Usage**

```r
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 = TRUE, threads = quanteda_options("threads"), 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
- `dispersion_floor` constraint for the minimal underdispersion multiplier in the quasi-Poisson model. Used to minimize the distorting effect of terms with rare term or document frequencies that appear to be severely underdispersed. Default is 0, but this only applies if `dispersion = "quasipoisson"`. 

---
sparse specifies whether the "dfm" is coerced to dense
threads specifies the number of threads to use; set to 1 to override the package settings and use a serial version of the function
abs_err specifies how the convergence is considered
svd_sparse uses svd to initialize the starting values of theta, only applies when sparse = TRUE
residual_floor specifies the threshold for residual matrix when calculating the svds, only applies when sparse = TRUE

Details
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(σ) = 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
Examples

textmodel_wordfish(data_dfm_lbgexample, dir = c(1,5))

## Not run:
ie2010dfm <- dfm(data_corpus_irishbudget2010, verbose = FALSE)
(wfm1 <- textmodel_wordfish(iie2010dfm, dir = c(6,5)))
(wfm2a <- textmodel_wordfish(iie2010dfm, dir = c(6,5),
dispersion = "quasipoisson", dispersion_floor = 0))
(wfm2b <- textmodel_wordfish(iie2010dfm, dir = c(6,5),
dispersion = "quasipoisson", dispersion_floor = .5))
plot(wfm2a@phi, wfm2b@phi, xlab = "Min underdispersion = 0", ylab = "Min underdispersion = .5",
    xlim = c(0, 1.0), ylim = c(0, 1.0))
plot(wfm2a@phi, wfm2b@phi, xlab = "Min underdispersion = 0", ylab = "Min underdispersion = .5",
    xlim = c(0, 1.0), ylim = c(0, 1.0), type = "n")
underdispersedTerms <- sample(which(wfm2a@phi < 1.0), 5)
which(leafnames(iie2010dfm) %in% names(topfeatures(iie2010dfm, 20)))
text(wfm2a@phi, wfm2b@phi, wfm2a@features,
    cex = .8, xlim = c(0, 1.0), ylim = c(0, 1.0), col = "grey90")
text(wfm2a@phi[underdispersedTerms], wfm2b@phi[underdispersedTerms],
    wfm2a@features[underdispersedTerms],
    cex = .8, xlim = c(0, 1.0), ylim = c(0, 1.0), col = "black")
if (require(austin)) {
wmodelaustin <- austin::wordfish(quanteda::as.wfm(iie2010dfm, dir = c(6,5))
cor(wfm1@theta, wmodelaustin$theta)
}
## End(Not run)

textmodel_wordscores    Wordscores text model

Description


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 for the to match the LBG (2003) method.
Details

Fitting a `textmodel_wordscores` results in an object of class `textmodel_wordscores_fitted` containing the following slots:

Slots

- `scale` linear or logit, according to the value of `scale`
- `Sw` the scores computed for each word in the training set
- `x` the dfm on which the wordscores model was called
- `y` the reference scores
- `call` the function call that fitted the model
- `method` takes a value of `wordscores` for this model

Predict Methods

A `predict` method is also available for a fitted wordscores object, see `predict.textmodel_wordscores_fitted`.

Author(s)

Kenneth Benoit

References


See Also

- `predict.textmodel_wordscores_fitted`

Examples

```r
(ws <- textmodel_wordscores(data_dfm_lbgexample, c(seq(-1.5, 1.5, .75), NA)))
predict(ws)
predict(ws, rescaling = "mv")
predict(ws, rescaling = "lbg")
```
Description

Estimate Lauderdale and Herzog’s (2016) model for one-dimensional document author (e.g. speakers) positions based on multiple groups of texts (e.g. debates). Each group of texts is scaled using Slapin and Proksch’s (2008) "wordfish" Poisson scaling model of one-dimensional document positions, and then the positions from a particular author are scaled across groups using a second-level linear factor model, using conditional maximum likelihood.

Usage

textmodel_wordshoal(x, groups, authors, dir = c(1, 2), tol = 0.001)

Arguments

x the dfm from which the model will be fit
groups the name of a variable in the document variables for data giving the document group for each document
authors the name of a variable in the document variables for data giving the author of each document
dir set global identification by specifying the indexes for a pair of authors such that \(\hat{\theta}_{dir[1]} < \hat{\theta}_{dir[2]}\)
tol a convergence threshold for the log-posterior of the model

Details

Returns estimates of relative author positions across the full corpus of texts.

Value

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

tol log-posterior tolerance used in fitting
dir global identification of the dimension
theta estimated document positions
beta debate marginal effects
alpha estimated document fixed effects
psi estimated document debate-level positions
groups document groups
authors document authors
ll log likelihood at convergence
se.theta standard errors for theta-hats
data corpus to which the model was fit
Author(s)
Benjamin Lauderdale and Kenneth Benoit

References

Examples
```r
## Not run:
data(data_corpus_irish30, package = "quantedaData")
iedfm <- dfm(data_corpus_irish30, remove_punct = TRUE)
wordshoalfit <-
textmodel_wordshoal(iedfm, dir = c(7,1),
    groups = docvars(data_corpus_irish30, "debateID"),
    authors = docvars(data_corpus_irish30, "member.name"))
fitdf <- merge(as.data.frame(summary(wordshoalfit)),
    docvars(data_corpus_irish30),
    by.x = "row.names", by.y = "member.name")
fitdf <- subset(fitdf, !duplicated(memberID))
aggregate(theta ~ party.name, data = fitdf, mean)
## End(Not run)
```

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, n = 20L, min_count = 2L)
```

Arguments
- **x**: a return object from `textstat_keyness`
- **show_reference**: logical; if TRUE, show key reference features in addition to key target features
- **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

Value
- a `ggplot2` object
textplot_scale1d

Author(s)
Haiyan Wang

See Also
textstat_keyness

Examples

## Not run:
# compare Trump v. Obama speeches
prescorpus <- corpus_subset(data_corpus_inaugural,
   President %in% c("Obama", "Trump"))
presdfm <- dfm(prescorpus, groups = "President", remove = stopwords("english"),
   remove_punct = TRUE)
result <- textstat_keyness(presdfm, target = "Trump")

# plot estimated word keyness
textplot_keyness(result)
textplot_keyness(result, show_reference = FALSE)

## End(Not run)

---

textplot_scale1d

plot a fitted scaling model

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, highlighted = NULL, alpha = 0.7,
   highlighted_color = "black")

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, coef.textmodel

Examples

## Not run:
```
 ie_dfm <- dfm(data_corpus_irishbudget2010)
doclab <- apply(docvars(data_corpus_irishbudget2010, c("name", "party")),
                 1, paste, collapse = " ")

## wordscores
refscores <- c(rep(NA, 4), -1, 1, rep(NA, 8))
ws <- textmodel(ie_dfm, refscores, model="wordscores", smooth = 1)
pred <- predict(ws)
# plot estimated word positions
textplot_scale1d(pred, margin = "features",
                 highlighted = c("minister", "have", "our", "budget"))
# plot estimated document positions
textplot_scale1d(pred, margin = "documents",
                 doclabels = doclab,
                 groups = docvars(data_corpus_irishbudget2010, "party"))
```

## wordfish
textplot_wordcloud

wfm <- textmodel_wordfish(dfm(data_corpus_irishbudget2010), dir = c(6,5))
# plot estimated document positions
textplot_scale1d(wfm, doclabels = doclab)
textplot_scale1d(wfm, doclabels = doclab,
                 groups = docvars(data_corpus_irishbudget2010, "party"))
# plot estimated word positions

## correspondence analysis
wca <- textmodel_ca(ie_dfm)
# plot estimated document positions

textplot_scale1d(wca, margin = "documents",
                 doclabels = doclab,
                 groups = docvars(data_corpus_irishbudget2010, "party"))

## End(Not run)

---

**textplot_wordcloud**

*plot features as a wordcloud*

**Description**

Plot a dfm or tokens 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**

```r
textplot_wordcloud(x, comparison = FALSE, ...)
```

**Arguments**

- **x**: a dfm object
- **comparison**: if TRUE, plot a `comparison.cloud` instead of a simple wordcloud, one grouping per document
- **...**: additional parameters passed to `wordcloud` or to `text` (and `strheight`, `strwidth`)

**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 or tokens 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.
See Also

wordcloud, comparison.cloud

Examples

# plot the features (without stopwords) from Obama's two inaugural addresses
mydfm <- dfm(corpus_subset(data_corpus_inaugural, President=="Obama"), verbose = FALSE,
             remove = stopwords("english"))
textplot_wordcloud(mydfm)

# plot in colors with some additional options passed to wordcloud
textplot_wordcloud(mydfm, random.color = TRUE, rot.per = .25,
                   colors = sample(colors()[2:128], 5))

## Not run:
# comparison plot of Irish government vs opposition
docvars(data_corpus_irishbudget2010, "govtopp") <-
   factor(ifelse(data_corpus_irishbudget2010[,"party"] %in% c("FF", "Green", "Govt", "Opp"))
govtoppdfm <- dfm(data_corpus_irishbudget2010, groups = "govtopp", verbose = FALSE)
textplot_wordcloud(tfidf(govtoppdfm), comparison = TRUE)
# compare to non-tf-idf version
textplot_wordcloud(govtoppdfm, comparison = TRUE)

## End(Not run)

---

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

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

Author(s)

Adam Obeng

Examples

```r
## Not run:
data_corpus_inauguralPost70 <- corpus_subset(data_corpus_inaugural, Year > 1970)
# compare multiple documents
textplot_xray(kwic(data_corpus_inauguralPost70, "american"))
textplot_xray(kwic(data_corpus_inauguralPost70, "american"), scale = "absolute")
# compare multiple terms across multiple documents
textplot_xray(kwic(data_corpus_inauguralPost70, "america"),
              kwic(data_corpus_inauguralPost70, "people"))

# how to modify the ggplot with different options
library(ggplot2)
g <- textplot_xray(kwic(data_corpus_inauguralPost70, "american"),
                   kwic(data_corpus_inauguralPost70, "people"))
g + aes(color = keyword) + scale_color_manual(values = c('red', 'blue'))
## End(Not run)
```

Description

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

Usage

```r
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.
when concatenating texts by using groups, this will be the spacing added between texts. (Default is two spaces.)

character vector of the new texts

unused

\textit{Details}

\texttt{as.character(x) where x is a corpus is equivalent to calling \texttt{texts(x)}}

\textit{Value}

For \texttt{texts}, a character vector of the texts in the corpus.

For \texttt{texts <-}, the corpus with the updated texts.

For \texttt{texts <-, a corpus with the texts replaced by value}

\texttt{as.character(x) is equivalent to \texttt{texts(x)}}

\textit{Note}

The \texttt{groups} will be used for concatenating the texts based on shared values of \texttt{groups}, without any specified order of aggregation.

You are strongly encouraged as a good practice of text analysis workflow \textit{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 \texttt{tokens_tolower} after applying \texttt{tokens} to a corpus, or use the option \texttt{tolower = TRUE} in \texttt{dfm}.

\textit{Examples}

\begin{verbatim}
 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)
\end{verbatim}
**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 $c(x, z) = (j_1, \ldots, j_K) = c$ such that the $k$th element of $c$ is 1 if the $k$th word in $z$ is equal
to the \( k \)th word in \( x \), and 0 otherwise. Let \( c_i = (j_{i1}, \ldots, j_{iK}), i = 1, \ldots, 2^K = M \), be the possible values of \( c(x, z) \), with \( c_M = (1, 1, \ldots, 1) \). Consider the set of \( c(x, z_r) \) across all expressions \( z_r \) in a corpus of text, and let \( n_i \), for \( i = 1, \ldots, M \), denote the number of the \( c(x, z_r) \) which equal \( c_i \), plus the smoothing constant \( \text{smoothing} \). The \( n_i \) are the counts in a \( 2^K \) contingency table whose dimensions are defined by the \( c_i \).

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

\[
\lambda = \sum_{i=1}^{M} (-1)^{K-b_i} \log n_i
\]

where \( b_i \) is the number of the elements of \( c_i \) 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.
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 next release of quanteda.

Author(s)

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

References


Examples

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

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

Similarity and distance computation between documents or features

Description

These functions compute matrixes of distances and similarities between documents or features from a dfm and return a dist object (or a matrix if specific targets are selected). They are fast and robust because they operate directly on the sparse dfm objects.

Usage

textstat_dist(x, selection = NULL, margin = c("documents", "features"),
method = "euclidean", upper = FALSE, diag = FALSE, p = 2)
textstat_simil(x, selection = NULL, margin = c("documents", "features"),
method = "correlation", upper = FALSE, diag = FALSE)

Arguments

x a dfm object

selection character vector of document names or feature labels from x. A "dist" object is returned if selection is NULL, otherwise, a matrix is returned.

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

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

upper whether the upper triangle of the symmetric \( V \times V \) matrix is recorded

diag whether the diagonal of the distance matrix should be recorded

p The power of the Minkowski distance.

Details

textstat_dist options are: "euclidean" (default), "Chisquared", "Chisquared2", "hamming", "kullback", "manhattan", "maximum", "canberra", and "minkowski".
textstat_simil options are: "correlation" (default), "cosine", "jaccard", "eJaccard", "dice", "eDice", "simple matching", "hamann", and "faith".

Value

textstat_simil and textstat_dist return dist class objects.

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, "relfreq").
Author(s)
Kenneth Benoit, Haiyan Wang

References


"hamming" is $\sum x \neq y$.

"kullback" is the Kullback-Leibler distance, which assumes that $P(x_i) = 0$ implies $P(y_i) = 0$, and in case both $P(x_i)$ and $P(y_i)$ equals to zero, then $P(x_i) * log(p(x_i)/p(y_i))$ is assumed to be zero as the limit value. The formula is:

$$\sum P(x) * log(P(x)/p(y))$$

All other measures are described in the *proxy* package.

See Also
textstat_dist.as.list.dist.dist

Examples

```r
# create a dfm from inaugural addresses from Reagan onwards
presDfm <- dfm(corpus_subset(data_corpus_inaugural, Year > 1990),
                remove = stopwords("english"), stem = TRUE, remove_punct = TRUE)

# distances for documents
(d1 <- textstat_dist(presDfm, margin = "documents"))
as.matrix(d1)

# distances for specific documents
textstat_dist(presDfm, "2017-Trump", margin = "documents")
textstat_dist(presDfm, "2005-Bush", margin = "documents", method = "eJaccard")
(d2 <- textstat_dist(presDfm, c("2009-Obama", "2013-Obama"), margin = "documents"))
as.list(d2)

# similarities for documents
(s1 <- textstat_simil(presDfm, method = "cosine", margin = "documents"))
as.matrix(s1)
as.list(s1)

# similarities for specific documents
textstat_simil(presDfm, "2017-Trump", margin = "documents")
textstat_simil(presDfm, "2017-Trump", method = "cosine", margin = "documents")
textstat_simil(presDfm, c("2009-Obama", "2013-Obama"), margin = "documents")
```
# compute some term similarities
s2 <- textstat_simil(presDfm, c("fair", "health", "terror"), method = "cosine", margin = "features")
head(as.matrix(s2), 10)
as.list(s2, n = 8)

textstat_frequency  tabulate feature frequencies

Description

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

Usage

```r
textstat_frequency(x, n = NULL, groups = NULL)
```

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.

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.
Examples

df1 <- dfm(c("a a b b c d", "a d d d", "a a a"))
textstat_frequency(dfm1)
textstat_frequency(dfm1, groups = c("one", "two", "one"))

obamadfm <-
corpus_subset(data_corpus_inaugural, President == "Obama") %>
    dfm(remove_punct = TRUE, remove = stopwords("english"))
freq <- textstat_frequency(obamadfm)
head(freq, 10)

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

# plot relative frequencies by group
dfm_weight_pres <- data_corpus_inaugural %>
    corpus_subset(Year > 2000) %>
    dfm(remove = stopwords("english"), remove_punct = TRUE) %>
    dfm_weight(type = "relfreq")

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

# plot frequencies
    ggplot(data = freq_weight, aes(x = nrow(freq_weight):1, y = frequency)) +
    geom_point() +
    facet_wrap(~ group, scales = "free") +
    coord_flip() +
    scale_x_continuous(breaks = nrow(freq_weight):1,
        labels = freq_weight$feature) +
    labs(x = NULL, y = "Relative frequency")

---

textstat_keyness

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 dfm, with the reference group consisting of all other documents.
textstat_keyness

Usage

textstat_keyness(x, target = 1L, measure = c("chi2", "exact", "lr", "pmi"),
sort = TRUE, correction = c("default", "yates", "williams", "none"))

Arguments

x a dfm containing the features to be examined for keyness
target 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
measure (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.
sort logical; if TRUE sort features scored in descending order of the measure, otherwise leave in original feature order
correction 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.

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 $G_2$ statistic; for "pmi" this is the pointwise mutual information statistics.

References


Examples

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

# compare pre- v. post-war terms using logical vector
mydfm2 <- dfm(data_corpus_inaugural)
textstat_keyness(mydfm2, docvars(data_corpus_inaugural, "Year") >= 1945)

# compare Trump 2017 to other post-war presidents
pwdfm <- dfm(corpus_subset(data_corpus_inaugural, period == "post-war"))
head(textstat_keyness(pwdfm, target = "2017-Trump"), 10)
# using the likelihood ratio method
head(textstat_keyness(dfm_smooth(pwdfm), measure = "lr", target = "2017-Trump"), 10)
```

---

**textstat_lexdiv**

**calculate lexical diversity**

**Description**

Calculate the lexical diversity or complexity of text(s).

**Usage**

```r
textstat_lexdiv(x, measure = c("all", "TTR", "C", "R", "CTTR", "U", "S", "Maas"), log.base = 10, drop = TRUE, ...)
```

**Arguments**

- `x`: an input object, such as a document-feature matrix object
- `measure`: a character vector defining the measure to calculate.
- `log.base`: a numeric value defining the base of the logarithm (for measures using logs)
- `drop`: if TRUE, the result is returned as a numeric vector if only a single measure is requested; otherwise, a data.frame is returned with each column consisting of a requested measure.
- `...`: not used

**Details**

`textstat_lexdiv` calculates a variety of proposed indices for lexical diversity. In the following formulae, $N$ refers to the total number of tokens, and $V$ to the number of types:

"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 LogTTR):

$$ C = \frac{\log V}{\log N} $$
"R": Guiraud’s Root TTR (Guiraud, 1954, as cited in Tweedie & Baayen, 1998):
\[ R = \frac{V}{\sqrt{N}} \]

"CTTR": Carroll’s Corrected TTR:
\[ CTTR = \frac{V}{\sqrt{2N}} \]

"U": Dugast’s 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 cited in Tweedie & Baayen, 1998) is calculated by:
\[ K = 10^4 \times \frac{\left( \sum_{X=1}^{N} f_X X^2 \right) - N}{N^2} \]

where \( N \) is the number of tokens, \( X \) is a vector with the frequencies of each type, and \( f_X \) is the frequencies for each \( X \).

"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.

**Value**

a data.frame or vector of lexical diversity statistics, each row or vector element corresponding to an input document

**Note**

This implements only the static measures of lexical diversity, not more complex measures based on windows of text such as the Mean Segmental Type-Token Ratio, the Moving-Average Type-Token Ratio (Covington & McFall, 2010), the MLTD or MLTD-MA (Moving-Average Measure of Textual Lexical Diversity) proposed by McCarthy & Jarvis (2010) or Jarvis (no year), or the HD-D version of vocd-D (see McCarthy & Jarvis, 2007). These are available from the package korRpus.
Author(s)
Kenneth Benoit, adapted from the S4 class implementation written by Meik Michalke in the \texttt{koRpus} package.

References


Michalke, Meik. (2014) \texttt{koRpus: An R Package for Text Analysis}. Version 0.05-5. \url{http://reaktanz.de/c=hacking&s=koRpus}


Examples

```r
mydfm <- dfm(corpus_subset(data_corpus_inaugural, Year > 1980), verbose = FALSE)
(results <- textstat_lexdiv(mydfm, c("CTTR", "TTR", "U")))
cor(textstat_lexdiv(mydfm, "all"))

# with different settings of drop
textstat_lexdiv(mydfm, "TTR", drop = TRUE)
textstat_lexdiv(mydfm, "TTR", drop = FALSE)
```

**Description**

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

**Usage**

```r
calculate readability"
```

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

"Traenkle.Bailer", "Traenkle.Bailer.2", "Wheeler.Smith", "meanSentenceLength", "meanWordSyllables"), remove_hyphens = TRUE, min_sentence_length = 1, max_sentence_length = 10000, drop = TRUE, ...)

Arguments

x
measure
remove_hyphens
min_sentence_length, max_sentence_length
drop
...

Value

a data.frame object consisting of the documents as rows, and the readability statistics as columns

Author(s)

Kenneth Benoit, re-engineered from the function of the same name by Meik Michalke in the korpus package.

Examples

txt <- c("Readability zero one. Ten, Eleven.", "The cat in a dilapidated tophat.")
textstat_readability(txt, "Flesch.Kincaid")
textstat_readability(txt, "Flesch.Kincaid", drop = FALSE)
textstat_readability(txt, c("FOG", "FOG.PSK", "FOG.NRI"))
inaugReadability <- textstat_readability(data_corpus_inaugural, "all")
round(cor(inaugReadability), 2)

textstat_readability(data_corpus_inaugural, measure = "Flesch.Kincaid")
inaugReadability <- textstat_readability(data_corpus_inaugural, "all")
round(cor(inaugReadability), 2)
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 = FALSE, remove_separators = TRUE,
remove_twitter = FALSE, remove_hyphens = FALSE, remove_url = FALSE,
ngrams = 1L, skip = 0L, concatenator = ",", hash = TRUE,
verbose = quanteda_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{white_space}"}
"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
remove tokens that consist only of numbers, but not words that start with digits, e.g. 2day

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

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

remove_separators
remove Separators and separator characters (spaces and variations of spaces, plus tab, newlines, and anything else in the Unicode "separator" category) when remove_punct=FALSE. 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). Note that if what = "word" and you set remove_punct = TRUE, then remove_separators has no effect. Use carefully.

remove_twitter
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
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".
remove_url
  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

hash
  if TRUE (default), return a hashed tokens object, otherwise, return a classic tokenizedTexts object. (This will be phased out soon in coming versions.)

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.

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 hashed 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

txt <- c(doc1 = "This is a sample: of tokens.",
     doc2 = "Another sentence, to demonstrate how tokens works.")
tokens(txt)
# removing punctuation marks and lowecasing texts
tokens(char_tolower(txt), 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(txt, remove_url = TRUE, remove_punct = TRUE)
tokens(txt, remove_url = FALSE, remove_punct = TRUE)
tokens(txt, remove_url = FALSE, remove_punct = TRUE, what = "fasterword")
tokens(txt, remove_url = FALSE, remove_punct = FALSE, what = "fasterword")

## MORE COMPARISONS

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


txt <- c(text1="This is $10 in 999 different ways,\n up and down; left and right!",
     text2="@kenbenoit working: on #quanteda 2day\nt4ever, http://textasdata.com?page=123.")
tokens(txt, verbose = TRUE)
tokens(txt, remove_numbers = TRUE, remove_punct = TRUE)
tokens(txt, remove_numbers = FALSE, remove_punct = TRUE)
tokens(txt, remove_numbers = TRUE, remove_punct = FALSE)
tokens(txt, remove_numbers = FALSE, remove_punct = FALSE)
tokens(txt, remove_numbers = FALSE, remove_punct = FALSE, remove_separators = FALSE)
tokens(txt, 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.")
tokens_compound

"Today is Thursday in Canberra. \nIt is yesterday in London. ",
"To be? Or\nnot to be?\n",
what = "sentence"
)
tokens(data_corpus_inaugural[c(2,48)], what = "sentence")

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

# ngram tokenization
### tokens(txt, remove_punct = TRUE, ngrams = 2)
### tokens(txt, remove_punct = TRUE, ngrams = 2, skip = 1, concatenator = " ")
### tokens(txt, remove_punct = TRUE, ngrams = 1:2)
# removing features from ngram tokens
### removeFeatures(tokens(txt, remove_punct = TRUE, ngrams = 1:2), stopwords("english"))

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

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, collocations, or dfm. 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 valuety for details.
  case_insensitive logical; if TRUE, ignore case when matching
  join logical; if TRUE, join overlapping compounds
Value

A `tokens` object in which the token sequences matching pattern have been replaced by compound "tokens" joined by the concatenator

Author(s)

Kenneth Benoit and Kohei Watanabe

Examples

```r
mytexts <- c("The new law included a capital gains tax, and an inheritance tax.",
             "New York City has raised taxes: an income tax and inheritance taxes.")
mytoks <- tokens(mytexts, remove_punct = TRUE)

# for lists of sequence elements
myseqs <- list(c("tax"), c("income", "tax"), c("capital", "gains", "tax"), c("inheritance", "tax"))
(cw <- tokens_compound(mytoks, myseqs))
dfm(cw)

# when used as a dictionary for dfm creation
mydict <- dictionary(list(tax=c("tax", "income tax", "capital gains tax", "inheritance tax")))
(cw2 <- tokens_compound(mytoks, mydict))

# to pick up "taxes" in the second text, set valuetype = "regex"
(cw3 <- tokens_compound(mytoks, mydict, valuetype = "regex"))

# dictionaries w/glob matches
myDict <- dictionary(list(negative = c("bad* word*", "negative", "awful text"),
                         positive = c("good stuff", "like? th??")))
toks <- tokens(c(txt1 = "I liked this, when we can use bad words, in awful text.",
                txt2 = "Some damn good stuff, like the text, she likes that too."))
tokens_compound(toks, myDict)

# with collocations
cols <-
textstat_collocations(tokens("capital gains taxes are worse than inheritance taxes"),
            size = 2, min_count = 1)
toks <- tokens("The new law included capital gains taxes and inheritance taxes.")
tokens_compound(toks, cols)
```

tokens_lookup

Apply a dictionary to a tokens object

Description

Convert tokens into equivalence classes defined by values of a dictionary object.
Usage

tokens_lookup(x, dictionary, levels = 1:5, valuetype = c("glob", "regex", "fixed"), case_insensitive = TRUE, capkeys = !exclusive, exclusive = TRUE, nomatch = NULL, 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.
verbose print status messages if TRUE

Examples

toks <- tokens(data_corpus_inaugural)
dict <- dictionary(list(country = "united states",
law=c('law*', 'constitution'),
freedom=c('free*', 'libert*'))) dfm(tokens_lookup(toks, dict, valuetype='glob', verbose = TRUE)) dfm(tokens_lookup(toks, dict, valuetype='glob', verbose = TRUE, nomatch = 'NONE'))

dict_fix <- dictionary(list(country = "united states",
law = c('law', 'constitution'),
freedom = c('freedom', 'liberty'))) # dfm(applyDictionary(toks, dict_fix, valuetype='fixed')) dfm(tokens_lookup(toks, dict_fix, 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.")
toks <- tokens(txt)
dict <- dictionary(list(US = list(Countries = c("States"),


tokens_ngrams

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

tokens_ngrams(x, n = 2L, skip = 0L, concatenator = "_")

c_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 `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

```r
# 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 = " ")
```
tokens_select

select or remove tokens from a tokens object

Description

This function selects or discards tokens from a tokens objects, with the shortcut tokens_remove(x, pattern) defined as a shortcut for tokens_select(x, pattern, selection = "remove"). 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, verbose = quanteda_options("verbose"))

tokens_remove(x, pattern, valuetype = c("glob", "regex", "fixed"),
  case_insensitive = TRUE, padding = FALSE,
  verbose = quanteda_options("verbose"))

Arguments

- **x** tokens object whose token elements will be selected
- **pattern** a character vector, list of character vectors, dictionary, collocations, or dfm. 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 valuetype for details.
- **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.
- **verbose** if TRUE print messages about how many tokens were selected or removed

Value

a tokens object with tokens selected or removed based on their match to pattern
Examples

```r
## 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)

## tokens_remove example
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(toks, remove_punct = TRUE), stopwords("english"))
```

tokens_tolower convert the case of tokens

Description

tokens_tolower and tokens_toupper convert the features of a tokens object and reindex 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)

...

additional arguments passed to stringi functions, (e.g. stri_trans_tolower),
such as locale

Examples

```r
# 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_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](#)
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

```r
mydfm <- dfm(corpus_subset(data_corpus_inaugural, Year > 1980), remove_punct = TRUE)
mydfm_nostopw <- dfm_remove(mydfm, stopwords("english"))

# most frequent features
topfeatures(mydfm)
topfeatures(mydfm_nostopw)

# least frequent features
topfeatures(mydfm_nostopw, decreasing = FALSE)

# top features of individual documents
topfeatures(mydfm_nostopw, n = 5, groups = docnames(mydfm_nostopw))

# grouping by president last name
topfeatures(mydfm_nostopw, n = 5, groups = "President")

# features by document frequencies
tail(topfeatures(mydfm, scheme = "docfreq", n = 200))
```
Index

*Topic **bootstrap**
  bootstrap_dfm, 11

*Topic **character**
  corpus_segment, 20

*Topic **collocations**
  textstat_collocations, 81

*Topic **corpus**
  corpus, 15
  corpus_reshape, 18
  corpus_sample, 19
  corpus_segment, 20
  corpus_subset, 22
  dfm_subset, 38
  docnames, 46
  docvars, 47
  head.corpus, 51
  metacorpus, 55
  metadoc, 56
  texts, 79

*Topic **data**
  data_char_sampletext, 23
  data_char_ukimmig2010, 24
  data_corpus_inaugural, 24
  data_corpus_irishbudget2010, 25
  data_dfm_lbgexample, 26
  data_dictionary_lsdRP1U, 27

*Topic **dfm**
  as.matrix.dfm, 8
  bootstrap_dfm, 11
  dfm, 28
  dfm_lookup, 33
  dfm_select, 35
  dfm_weight, 42
  docnames, 46
  head_dfm, 52

*Topic **experimental**
  bootstrap_dfm, 11
  textmodel_wordshoal, 73
  textstat_collocations, 81

*Topic **plot**
  textplot_wordcloud, 77
  textplot_xray, 78
  textstat_frequency, 85

*Topic **textmodel**
  textmodel_wordshoal, 73

*Topic **textplot**
  textplot_keyness, 74
  textplot_scale1d, 75

*Topic **textstat**
  textstat_collocations, 81
  textstat_keyness, 86

*Topic **tokens**
  tokens, 92
  .tokens (as.tokens), 9
  ., 45
  [, 45
  [., 45

  as.character.corpus (texts), 79
  as.character.tokens (as.tokens), 9
  as.corpus.corpuszip, 6
  as.data.frame.dfm, 53
  as.data.frame.dfm (as.matrix.dfm), 8
  as.dfm (is.dfm), 53
  as.dictionary, 6, 45
  as.list, 45
  as.list.dist, 7, 84
  as.list.tokens, 94
  as.list.tokens (as.tokens), 9
  as.matrix.dfm, 8, 53
  as.tokens, 9
  as.tokens.kwic (kwic), 54
  as.yaml, 11

  bootstrap_dfm, 11, 36
  c.tokens, 10
  c.tokens (as.tokens), 9
  ca, 66
cbind.dfm, 31
char_ngrams (tokens_ngrams), 98
char_segment (corpus_segment), 20
chartolower, 12
chartolower (chartolower), 12
charwordstem (tokens_wordstem), 102
character, 15, 61, 102
coeff, 13
coeff.textmodel, 13, 76
coefficients, 13
collocations, 20, 36, 54, 55, 95, 100
corpus-class, 17
corpusreshape, 18, 21, 22
corpus_sample, 19
corpus_segment, 20, 21
corpussubset, 22
corpustrim, 91
data.frame, 15
data_char_samplerawtext, 23
data_char_stopwords, 65
data_char_ukimmig2010, 24
data_corpus_inaugural, 24
data_corpus_irishbudget2010, 25
data_dfm_LBGexample
  (data_dfm_lbgexample), 26
data_dfm_lbgexample, 26
data_dictionary_LSD2015, 27
descriptive statistics on text, 5
dfm, 5, 8, 11, 12, 14, 20, 26, 28, 29, 31, 32, 35–39, 41, 42, 45–48, 51–54, 57, 61, 66, 67, 71, 73, 77, 80, 83, 85–87, 93, 95, 100, 102, 103
dfm-class, 29
dfmcompress, 31
dfmgroup, 32
dfmlookup, 10, 29, 33
dfmremove (dfm_select), 35
dfmsample, 34, 41
dfmsel, 29, 35, 39, 41
dfmsmooth (dfm_weight), 42
dfmsort, 37
dfmsubset, 38
dfmtolower, 31, 39
dfmtolower (dfmtolower), 39
dfmsort, 36, 40
dfmsort (dfm_sort), 37
dfmsort (dfm_sort), 37
dfmweight, 42, 83
dfmwordstem (tokens_wordstem), 102
dictionaries, 5
dictionary, 7, 11, 20, 27, 29, 33, 36, 44, 54, 55, 95, 97, 100
dist, 83, 84
docfreq, 42, 43
docnames, 17, 31, 46
docnames <- (docnames), 46
document-feature matrix, 88
DocumentTermMatrix, 14
docvars, 15, 17, 23, 31, 32, 38, 47, 56, 85
docvars <- (docvars), 47
fcm, 31, 35, 36, 40, 48, 49, 50
fcmcompress (dfm_compress), 31
fcmremove (dfm_select), 35
fcmselect (dfm_select), 35
fcmsort, 50, 50
fcmtolower (dfmtolower), 39
fcmtolower (dfmtolower), 39
featnames, 31, 41, 46, 51
file, 44
getStemLanguages, 102
groups, 29, 32, 76, 79, 85, 103
head.corpus, 51
head.dfm, 52
iconv, 44
is.collocations
  (textstat_collocations), 81
is.dfm, 53
is.dictionary, 45
is.dictionary (as.dictionary), 6
is.fcm (fcm), 48
is.kwic (kwic), 54
is.phrase (phrase), 61
is.tokens (as.tokens), 9
key-wordsin-context, 5
keywords, 5
kwic, 15, 54, 78
lda.collapsed.gibbs.sampler, 14
lexical diversity measures, 5
list, 45
INDEX

metacorpus, 15, 17, 55, 56
metacorpus<- (metacorpus), 55
metadoc, 17, 56
metadoc<- (metadoc), 56

ndoc, 17, 57
nfeature, 49
nfeature (ndoc), 57
ngrams, 99
nscribble, 58
nsentence, 59
nsyllable, 59
ntoken, 57, 60
ntype (ntoken), 60

options, 62

pattern, 20, 28, 29, 36, 54, 61, 95, 100
pattern matches, 27
phrase, 55, 61, 62
predict.textmodel_NB_fitted, 68
predict.textmodel_wordscores_fitted, 72

quanteda (quanteda-package), 4
quanteda-package, 4, 25
quanteda_options, 62

rbind.dfm, 31

readability indexes, 5

sample, 19, 35
settings, 17
similarities, 5
SimpleCorpus, 15
spacyr-methods, 64
sparsity, 64
stopwords, 29, 65
strheight, 77
stri_opts_brkiter, 59
stri_split_charclass, 92
stri_split_fixed, 92
stri_trans_tolower, 12, 40, 101
stringi-search-boundaries, 92
strwidth, 77
subset, 17
subset.data.frame, 23, 38, 39
summary.corpus, 16
svds, 67
tail.corpus (head.corpus), 51
tail.dfm (head.dfm), 52
text, 77
textmodel_ca, 66, 75
textmodel_NB, 67
textmodel_wordfish, 69, 75, 76
textmodel_wordscores, 71, 75, 76
textmodel_wordshoal, 73
textplot_keyness, 74
textplot_scaleid, 75
textplot_wordcloud, 77
textplot_xray, 78
texts, 17, 79
texts<- (texts), 79
textstat_collocations, 81
textstat_dist, 7, 83, 84
textstat_frequency, 85
textstat_keyness, 74, 75, 86
textstat_lexdiv, 88
textstat_readability, 90
textstat_simil, 7
textstat_simil (textstat_dist), 83
tf, 42, 43, 68
tfidf, 42, 43
tokenize, 28
tokens, 9, 10, 18, 28, 29, 46–48, 54, 57,
      59–61, 77, 80, 81, 92, 92, 93, 95, 96,
      99–102
tokens_compound, 95
tokens_lookup, 10, 29, 34, 96
tokens_ngrams, 93, 94, 98
tokens_remove, 28, 81
tokens_remove (tokens_select), 100
tokens_select, 29, 100
tokens_skipgrams, 93, 94, 99
tokens_skipgrams (tokens_ngrams), 98
tokens_tolower, 80, 101
tokens_toupper (tokens_tolower), 101
tokens_wordstem, 102
tolower, 12
topfeatures, 103
toupper, 12

unlist, 10
unlist.tokens (as.tokens), 9

valuetype, 20, 29, 33, 36, 44, 54, 95, 97, 100
VCorpus, 15
wordcloud, 77, 78
wordStem, 102