Package 'quanteda'

April 20, 2017

Version 0.9.9-50

Title Quantitative Analysis of Textual Data

Description A fast, flexible framework for for the management, processing, and quantitative analysis of textual data in R.

License GPL-3

Depends R (>= 3.2.2), methods

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

LinkingTo Rcpp, RcppParallel, RcppArmadillo (>= 0.7.600.1.0)

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

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' 'dictionaries.R' 'corpus.R' 'collocations.R' 'collocations2.R' 'convert.R' 'corpus-deprecated.R' 'corpus-methods-base.R' 'corpus-methods-quanteda.R' 'corpus_reshape.R' 'corpus_sample.R' 'corpus_segment.R' 'corpus_subset.R' 'corpus_trimsentences.R' 'corpuszip.R' 'data-deprecated.R' 'data-documentation.R' 'dfm-classes.R' 'dfm-deprecated.R' 'dfm-methods.R' 'dfm-print.R' 'dfm-subsetting.R' 'dfm.R' 'dfm_compress.R' 'dfm_lookup.R' 'dfm_sample.R' 'dfm_select.R' 'dfm_trim.R' 'dfm_weight.R' 'dictionaries-deprecated.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.R' 'quanteda_options.R'

'readtext-methods.R' 'regex2fixed.R' 'resample.R'
'selectFeatures-old.R' 'selectFeatures.R' 'sequences.R'
'settings.R' 'similarity.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_scale1d.R' 'textplot_wordcloud.R' 'textplot_xray.R'
'textstat-deprecated.R' 'textstat_collocations.R'
'textstat_dist.R' 'textstat_keyness.R' 'textstat_lexdiv.R'
'textstat_readability.R' 'textstat_simil.R' 'toLower.R'
'tokenize.R' 'tokenize_outtakes.R' 'tokens.R'
'tokens_compound.R' 'tokens_lookup.R' 'tokens_ngrams.R'
'tokens_select.R' 'tolower-misc.R' 'utils.R' 'valuetype.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 2017-04-20 09:07:57 UTC

R topics documented:

`quanteda-package` *An R package for the quantitative analysis of textual data*

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 manuipulate the texts in a corpus, by performing the most common natural language processing tasks simply and quickly, such as tokenizing, stemming, or forming ngrams. **quanteda**'s functions for tokenizing texts and forming multiple tokenized documents into a document-feature matrix are both extremely fast and extremely simple to use. **quanteda** can segment texts easily by words, paragraphs, sentences, or even user-supplied delimiters and tags.

Built on the text processing functions in the **stringi** package, which is in turn built on C++ implementation of the ICU libraries for Unicode text handling, **quanteda** pays special attention to fast and correct implementation of Unicode and the handling of text in any character set.

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

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

Once constructed, a **quanteda** "dfm"" can be easily analyzed using either **quanteda**'s built-in tools for scaling document positions, or used with a number of other text analytic tools, such as: topic

models (including converters for direct use with the topicmodels, LDA, and stm packages) document scaling (using **quanteda**'s own functions for the "wordfish" and "Wordscores" models, direct use with the ca package for correspondence analysis, or scaling with the austin package) machine learning through a variety of other packages that take matrix or matrix-like inputs.

Additional features of **quanteda** include:

- the ability to explore texts using key-words-in-context;
- fast computation of a variety of readability indexes;
- fast computation of a variety of lexical diversity measures;
- quick computation of word or document similarities, for clustering or to compute distances for other purposes; and
- a comprehensive suite of descriptive statistics on text such as the number of sentences, words, characters, or syllables per document.

Source code and additional information

http://github.com/kbenoit/quanteda

Author(s)

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

Other contributors:

- Kohei Watanabe <watanabe.kohei@gmail.com> [contributor]
- Paul Nulty <paul.nulty@gmail.com> [contributor]
- Adam Obeng <quanteda@binaryeagle.com> [contributor]
- Haiyan Wang <h.wang52@lse.ac.uk> [contributor]
- Benjamin Lauderdale <B.E.lauderdale@lse.ac.uk> [contributor]
- Will Lowe <wlowe@princeton.edu> [contributor]

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

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

Arguments

x	a compressed corpus object

as.list.dist *coerce a dist object into a list*

Description

Coerce a dist matrix into a list of selected terms and tarhet terms in descending order. Can be used after calling textstat_simil or textstat_dist.

Usage

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

Arguments

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

Examples

```
## 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 <- new("dfmSparse", Matrix::Matrix(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)
```

`as.matrix.dfm` *coerce a dfm to a matrix or data.frame*

Description

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

Usage

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

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

Arguments

`x`	dfm to be coerced
`...`	not used
`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.
`optional`	not applicable to this method

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)
```

`as.tokens.collocations` *coercion and checking functions for tokens objects*

Description

Coerce a list of character elements to a quanteda tokens object, or check whether an object is a tokens object.

Usage

```
## S3 method for class 'collocations'
as.tokens(x)

as.tokens(x)

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

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

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

is.tokens(x)

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

## S3 method for class 'tokens'
c(..., recursive = FALSE)
```

Arguments

`x`	list of character elements
`...`	unused
`use.names`	logical; preserve names if `TRUE`. For `as.character` only.
`t1`	tokens one to be added
`t2`	tokens two to be added
`recursive`	logical used by ‘c()‘ method, always set to ‘FALSE‘

Value

as.tokens returns a quanteda tokens object

as.list returns a simple list of characters from a tokens object

as.character returns a character vector from a tokens object

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

Examples

```
toks1 <- tokens(data_corpus_inaugural[1:5])
toks2 <- tokens(data_corpus_inaugural[21:25])
toks3 <- toks1 + toks2


toks1 <- tokens(data_corpus_inaugural[1:5])
toks2 <- tokens(data_corpus_inaugural[21:25])
toks3 <- tokens(data_corpus_inaugural[41:45])
summary(c(toks1, toks2, toks3))
```

---

as.yaml *convert quanteda dictionary objects to the YAML format*

Description

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

Usage

```
as.yaml(x)
```

Arguments

x	dictionary object

Value

as.yaml returns a dictionary in the YAML format

Examples

```
## Not run:
dict <- dictionary(file = '/home/kohei/Documents/Dictionary/LaverGarry.txt', format = 'wordstat')
yaml <- as.yaml(dict)
cat(yaml, file = '/home/kohei/Documents/Dictionary/LaverGarry.yaml')

## End(Not run)
```

bootstrap_dfm *bootstrap a dfm*

Description

Create an array of resampled dfms.

Usage

```
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

This code loops the creation of a dfm from a corpus, keeping a constant set of features based on the original dfm. Resampling of the corpus is done at the sentence level, within document.

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

```
txt <- c(textone = "This is a sentence.  Another sentence.  Yet another.",
         texttwo = "Premiere phrase.  Deuxieme phrase.")
bootstrap_dfm(txt, n = 3)
```

char_tolower *convert the case of character objects*

Description

char_tolower and char_toupper are replacements for tolower and toupper based on the **stringi** package.

Usage

```
char_tolower(x, keep_acronyms = FALSE, ...)

char_toupper(x, ...)
```

Arguments

x	a tokens object
keep_acronyms	if TRUE, do not lowercase any all-uppercase words. Only applies to char_tolower.
...	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(text1 = "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)
```

coef.textmodel *extract text model coefficients*

Description

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

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

Description

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

Usage

```
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 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

```
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
ldadfm <- convert(quantdfm, to = "lda")
str(ldadfm)

## 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 readtext object, from the **readtext** package (which is a specially constructed data.frame)
- a data.frame, whose default variable containing the document is character vector named `text`, although this can be set to any other variable name using the `text_field` argument. Other variables are imported as document-level meta-data.
- a kwic object constructed by `kwic`.
- a **tm** VCorpus class object, with the fixed metadata fields imported as document-level meta-data. Corpus-level metadata is not currently imported.

Usage

```
corpus(x, docnames = NULL, docvars = NULL, text_field = "text",
  metacorpus = NULL, compress = FALSE, ...)
```

Arguments

`x`	a valid corpus source object
`docnames`	Names to be assigned to the texts, defaults to the names of the character vector (if any), otherwise assigns "text1", "text2", etc.
`docvars`	A data frame of attributes that is associated with each text.
`text_field`	the character name or numeric index of the source `data.frame` indicating the variable to be read in as text, which must be a character vector. All other variables in the data.frame will be imported as docvars. This argument is only used for `data.frame` objects (including those created by **readtext**).
`metacorpus`	a named list containing additional (character) information to be added to the corpus as corpus-level metadata. Special fields recognized in the `summary.corpus` are: • `source` a description of the source of the texts, used for referencing; • `citation` information on how to cite the corpus; and • `notes` any additional information about who created the text, warnings, to do lists, etc.
`compress`	logical; if `TRUE`, compress the texts in memory using gzip compression. This significantly reduces the size of the corpus in memory, but will slow down operations that require the texts to be extracted.
`...`	not used directly

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)`. Because a corpus is also a list, it is also possible to access, create, or replace docvars using list notation, e.g.

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

For details, see corpus-class.

Value

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

A warning on accessing corpus elements

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

Author(s)

Kenneth Benoit and Paul Nulty

See Also

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

Examples

```
# create a corpus from texts
corpus(data_char_ukimmig2010)

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

corpus(texts(data_corpus_irishbudget2010))

# import a tm VCorpus
if ("tm" %in% rownames(installed.packages())) {
    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.")))

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

---

corpus_reshape *recast the document units of a corpus*

Description

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

Usage

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

Arguments

x	corpus whose document units will be reshaped
to	new document units in which the corpus will be recast
...	not used

Value

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

Examples

```
# simple example
mycorpus <- corpus(c(textone = "This is a sentence.  Another sentence.  Yet another.",
                     textwo = "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(mycorpus)
summary(corpus_reshape(mycorpus, to = "sentences"), showmeta=TRUE)

# example with inaugural corpus speeches
(mycorpus2 <- corpus_subset(data_corpus_inaugural, Year>2004))
paragCorpus <- corpus_reshape(mycorpus2, to="paragraphs")
paragCorpus
summary(paragCorpus, 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

Takes a random sample or documents or features 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

```
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

```
# 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 into component elements*

Description

Segment corpus text(s) or a character vector into tokens, sentences, paragraphs, or other sections. `segment` works on a character vector or corpus object, and allows the delimiters to be user-defined. This is useful for breaking the texts of a corpus into smaller documents based on sentences, or based on a user defined "tag" pattern. See details.

Usage

```
corpus_segment(x, what = c("sentences", "paragraphs", "tokens", "tags",
  "other"), delimiter = NULL, valuetype = c("regex", "fixed", "glob"),
  use_docvars = TRUE, ...)

char_segment(x, what = c("sentences", "paragraphs", "tokens", "tags",
  "other"), delimiter = NULL, valuetype = c("regex", "fixed", "glob"),
  use_docvars = TRUE, ...)
```

Arguments

`x`	character or corpus object whose texts will be segmented
`what`	unit of segmentation. Current options are `"sentences"` (default), `"paragraphs"`, `"tokens"`, `"tags"`, and `"other"`. Segmenting on `"other"` allows segmentation of a text on any user-defined value, and must be accompanied by the `delimiter` argument. Segmenting on `"tags"` performs the same function but preserves the tags as a document variable in the segmented corpus.
`delimiter`	delimiter defined as a `regex` for segmentation; only relevant for `what = "paragraphs"` (where the default is two newlines), `"tags"` (where the default is a tag preceded by two pound or "hash" signs ##), and `"other"`.
`valuetype`	how to interpret keyword expressions: `"glob"` for "glob"-style wildcard expressions; `"regex"` for regular expressions; or `"fixed"` for exact matching. See valuetype for details.
`use_docvars`	(for corpus objects only) 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.
`...`	provides additional arguments passed to `tokens`, if `what = "tokens"` is used

Details

Tokens are delimited by separators. For tokens and sentences, these are determined by the tokenizer behaviour in `tokens`.

For paragraphs, the default is two carriage returns, although this could be changed to a single carriage return by changing the value of `delimiter` to `"\\n{1}"` which is the R version of the regex for one newline character. (You might need this if the document was created in a word processor, for instance, and the lines were wrapped in the window rather than being hard-wrapped with a newline character.)

Value

`corpus_segment` returns a corpus of segmented texts, with a `tag` docvar if `what = "tags"`.

`corpus_segment` returns a character vector of segmented texts

Using delimiters

One of the most common uses for `corpus_segment` is to partition a corpus into sub-documents using tags. By default, the tag value is any word that begins with a double "hash" sign and is followed by a whitespace. This can be modified but be careful to use the syntax for the trailing word boundary (`\\b`)

The default values for `delimiter` are, according to valuetype:

paragraphs `"\\n{2}"`, regular expression meaning two newlines. If you wish to define a paragaph as a single newline, change the 2 to a 1.

tags `"##\\w+\\b"`, a regular expression meaning two "hash" characters followed by any number of word characters followed by a word boundary (a whitespace or the end of the text).

other No default; user must supply one.

tokens, sentences Delimiters do not apply to these, and a warning will be issued if you attempt to supply one.

Delimiters may be defined for different valuetypes but these may produce unexpected results, for example the lack of the ability in a "glob" expression to define the word boundaries.

Note

Does not currently record document segments if segmenting a multi-text corpus into smaller units. For this, use corpus_reshape instead.

Author(s)

Kenneth Benoit

See Also

`corpus_reshape`, `tokens`

Examples

```
## segmenting a corpus

testCorpus <-
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."))
# add a docvar
testCorpus[["serialno"]] <- paste0("textSerial", 1:ndoc(testCorpus))
testCorpusSeg <- corpus_segment(testCorpus, "tags")
summary(testCorpusSeg)
texts(testCorpusSeg)
# segment a corpus into sentences
segmentedCorpus <- corpus_segment(corpus(data_char_ukimmig2010), "sentences")
summary(segmentedCorpus)

## segmenting a character object

# same as tokenize()
identical(as.character(tokens(data_char_ukimmig2010)),
          as.character(char_segment(data_char_ukimmig2010, what = "tokens")))

# segment into paragraphs
char_segment(data_char_ukimmig2010[3:4], "paragraphs")

# segment a text into sentences
segmentedChar <- char_segment(data_char_ukimmig2010, "sentences")
segmentedChar[3]
```

`corpus_subset` *extract a subset of a corpus*

Description

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

Usage

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

Arguments

`x`	corpus object to be subsetted
`subset`	logical expression indicating elements or rows to keep: missing values are taken as false
`select`	expression, indicating the attributes to select from the corpus
`...`	not used

Value

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

See Also

subset.data.frame

Examples

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

---

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

Description

This is a long paragraph (2,914 characters) of text taken from an Irish budget speech by Joe Higgins.

Usage

```
data_char_sampletext
```

Format

character vector with one element

Examples

```
tokenize(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 docvars including year, and the last and first names of the presidents delivering the inaugural address

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.

References

https://archive.org/details/Inaugural-Address-Corpus-1789-2009 and http://www.presidency.ucsb.edu/inaugurals.php.

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

Lowe, Will, and Kenneth R Benoit. 2013. "Validating Estimates of Latent Traits From Textual Data Using Human Judgment as a Benchmark." *Political Analysis* 21: 298-313.

Examples

```
summary(data_corpus_irishbudget2010)
```

data_dfm_LBGexample *dfm from data in Table 1 of Laver, Benoit, and Garry (2003)*

Description

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

Usage

```
data_dfm_LBGexample
```

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

Laver, Michael, Kenneth Benoit, and John Garry. 2003. "Estimating policy positions from political text using words as data." *American Political Science Review* 97(2): 311-331.

`dfm` *create a document-feature matrix*

Description

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

Usage

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

Arguments

`x`	character, corpus, tokens, or dfm object
`tolower`	convert all tokens to lowercase
`stem`	if `TRUE`, stem words
`select`	a user supplied regular expression defining which 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 = "^@\\w+\\b"` would be the regular expression version of this matching pattern. The pattern matching type will be set by `valuetype`.
`remove`	a character vector 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`. For behaviour of `remove` with `ngrams > 1`, see Details.
`thesaurus`	A list of character vector "thesaurus" entries, in a dictionary list format, which operates as a dictionary but without excluding values not matched from the dictionary. Thesaurus keys are converted to upper case to create a feature label in the dfm, as a reminder that this was not a type found in the text, but rather the label of a thesaurus key. For more fine-grained control over this and other aspects of converting features into dictionary/thesaurus keys from pattern matches to values, you can use `dfm_lookup` after creating the dfm.
`dictionary`	A list of character vector dictionary entries, including regular expressions (see examples)

valuetype	how to interpret keyword expressions: "glob" for "glob"-style wildcard expressions; "regex" for regular expressions; or "fixed" for exact matching. See valuetype for details.
groups	character vector containing the names of document variables for aggregating documents; only applies when calling dfm on a corpus object. When x is a dfm object, groups provides a convenient and fast method of combining and refactoring the documents of the dfm according to the groups.
verbose	display messages if TRUE
...	additional arguments passed to tokens, for character and corpus

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

See Also

dfm_select, dfm-class

Examples

```
## for a corpus
corpus_post80inaug <- corpus_subset(data_corpus_inaugural, Year > 1980)
dfm(corpus_post80inaug)
dfm(corpus_post80inaug, tolower = FALSE)

# grouping documents by docvars in a corpus
dfm(corpus_post80inaug, groups = "President", verbose = TRUE)

# with English stopwords and stemming
dfm(corpus_post80inaug, 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_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 = "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 ha8ers 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` *compress a dfm or fcm by combining identical dimension elements*

Description

"Compresses" 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 may also occur after lower-casing or stemming the features of a dfm, but this should only be done in very rare cases (approaching never: it's better to do this *before* constructing the dfm.) It could also be needed after a `cbind.dfm` or `rbind.dfm` operation.

Usage

```
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

Note

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

Examples

```
mat <- rbind(dfm(c("b A A", "C C a b B"), tolower = FALSE, verbose = FALSE),
             dfm("A C C C C C", tolower = FALSE, verbose = 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_lookup *apply a dictionary to a dfm*

Description

Apply a dictionary to a dfm by looking up all dfm features for matches in a a set of dictionary values, and combine 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, 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 hierachical 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	how to interpret keyword expressions: "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
verbose	print status messages if TRUE

Note

dfm_lookup should not be used with dictionaries containing multi-word values, because dfm features will already have been fixed using a specific ngram value which may not match the multi-word structure of the dictionary.

Examples

```
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")
dfm_lookup(myDfm, myDict, valuetype = "fixed", case_insensitive = FALSE)
```

`dfm_sample` *randomly sample documents or features from a dfm*

Description

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

Usage

```
dfm_sample(x, size = ndoc(x), replace = FALSE, prob = NULL,
  what = 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.
`what`	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

```
myDfm <- dfm(data_corpus_inaugural[1:10])
dfm_sample(myDfm)[, 1:10]
dfm_sample(myDfm, replace = TRUE)[, 1:10]
dfm_sample(myDfm, what = "features")[1:10, ]
```

---

dfm_select *select features from a dfm or fcm*

Description

This function selects or discards features from a dfm or fcm, based on a pattern match with the feature names. 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

```
dfm_select(x, features = NULL, documents = NULL, selection = c("keep",
  "remove"), valuetype = c("glob", "regex", "fixed"),
  case_insensitive = TRUE, min_nchar = 1, max_nchar = 63,
  padding = FALSE, verbose = quanteda_options("verbose"), ...)

dfm_remove(x, features = NULL, documents = NULL, ...)

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

fcm_remove(x, features, ...)
```

Arguments

x	the dfm or fcm object whose features will be selected
features	one of: a character vector of features to be selected, a dfm whose features will be used for selection, or a dictionary class object whose values (not keys) will provide the features to be selected. For dfm objects, see details in the Value section below.
documents	select documents based on their document names. Works exactly the same as features.
selection	whether to keep or remove the features
valuetype	how to interpret keyword expressions: "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. These arguments are Ignored when padding is `TRUE`.

`padding`
: if `TRUE` features or documents not existing in `x` is added to dfm. This option is available only when selection is `keep` and valuetype is `fixed`.

`verbose`
: if `TRUE` print message about how many features were removed

`...`
: supplementary arguments passed to the underlying functions in `stri_detect_regex`

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 `features` is a dfm object and `padding` is `TRUE`, then the returned object will be identical in its feature set to the dfm supplied as the `features` argument. This means that any features in `x` not in `features` will be discarded, and that any features in found in the dfm supplied as `features` 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 `features` is a dfm object, then the following settings are always used: `padding = TRUE`, `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 a the document-feature matrix, use `dfm_trim`.

Examples

```
myDfm <- 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)
mydict <- dictionary(list(countries = c("United_States", "Sweden", "France"),
                          wordsEndingInY = c("by", "my"),
                          notintext = "blahblah"))
dfm_select(myDfm, mydict)
dfm_select(myDfm, mydict, case_insensitive = FALSE)
dfm_select(myDfm, c("s$", ".y"), selection = "keep", valuetype = "regex")
dfm_select(myDfm, c("s$", ".y"), selection = "remove", valuetype = "regex")
dfm_select(myDfm, stopwords("english"), selection = "keep", valuetype = "fixed")
dfm_select(myDfm, stopwords("english"), selection = "remove", valuetype = "fixed")

# select based on character length
```

```
dfm_select(myDfm, 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", padding = TRUE, verbose = TRUE))
setequal(featnames(dfm2), featnames(dfm3))

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
fcm_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

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

Arguments

`x`	Document-feature matrix created by `dfm`
`decreasing`	TRUE (default) if 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

```
dtm <- dfm(data_corpus_inaugural)
dtm[1:10, 1:5]
dfm_sort(dtm)[1:10, 1:5]
dfm_sort(dtm, decreasing = FALSE, "both")[1:10, 1:5]
```

---

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 to lower and upper case, respectively, and then recombine the counts.

Usage

```
dfm_tolower(x)

dfm_toupper(x)

fcm_tolower(x)

fcm_toupper(x)
```

Arguments

x	the dfm or fcm object

Details

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

Examples

```
# for a document-feature matrix
mydfm <- dfm(c("b A A", "C 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 *trim a dfm using frequency threshold-based feature selection*

Description

Returns a document by feature matrix reduced in size based on document and term frequency, usually in terms of a minimum 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 attributes of the features themselves – such as selecting features 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's (see trim from the austin package)

See Also

dfm_select, dfm_sample

Examples

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

# keep only words occuring >=10 times and in >=2 docs
dfm_trim(myDfm, min_count = 10, min_docfreq = 2)

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

# keep only words occuring <=10 times and in <=2 docs
dfm_trim(myDfm, max_count = 10, max_docfreq = 2)

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

# keep only words occuring at least 0.01 times and in >=2 documents
dfm_trim(myDfm, min_count = .01, min_docfreq = 2)

# keep only words occuring 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

```
dfm_weight(x, type = c("frequency", "relFreq", "relMaxFreq", "logFreq",
  "tfidf"), weights = NULL)

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 "tfidf" Term-frequency * inverse document frequency. For a full explanation, see, for example, http://nlp.stanford.edu/IR-book/html/htmledition/term-frequency-and-weighting-1.html. This implementation will not return negative values. For finer-grained control, call tfidf directly.
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 fatures. 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

Details

This converts a matrix from sparse to dense format, so may exceed memory requirements depending on the size of your input matrix.

Value

The dfm with weighted values.

Note

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

Author(s)

Paul Nulty and Kenneth Benoit

References

Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schutze. *Introduction to Information Retrieval*. Vol. 1. Cambridge: Cambridge University Press, 2008.

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(logTfDtm <- dfm_weight(dtm, type = "logFreq"))
head(tfidf(logTfDtm, normalize = FALSE))

#' # 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(..., file = NULL, format = NULL, concatenator = " ",
  tolower = TRUE, encoding = "auto")
```

Arguments

`...`	a named list of character vector dictionary entries, including valuetype pattern matches, and including multi-word expressions separated by `concatenator`. The argument may be an explicit list or named set of elements that can be turned into a list. See examples. This argument may be omitted if the dictionary is read from `file`.

`file`	file identifier for a foreign dictionary
`format`	character identifier for the format of the foreign dictionary. If not supplied, the format is guessed from the dictionary file's extension. Available options are: `"wordstat"` format used by Provalis Research's Wordstat software `"LIWC"` format used by the Linguistic Inquiry and Word Count software `"yoshikoder"` format used by Yoshikoder software `"lexicoder"` format used by Lexicoder `"YAML"` the standard YAML format
`concatenator`	the character in between multi-word dictionary values. This defaults to `"_"` except LIWC-formatted files, which defaults to a single space `" "`.
`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.

Value

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

References

Wordstat dictionaries page, from Provalis Research http://provalisresearch.com/products/content-analysis-software/wordstat-dictionary/.

Pennebaker, J.W., Chung, C.K., Ireland, M., Gonzales, A., & Booth, R.J. (2007). The development and psychometric properties of LIWC2007. [Software manual]. Austin, TX (www.liwc.net).

Yoshikoder page, from Will Lowe http://conjugateprior.org/software/yoshikoder/.

Lexicoder format, http://www.lexicoder.com

See Also

dfm

Examples

```
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))

# also works
mydict2 <- dictionary(christmas = c("Christmas", "Santa", "holiday"),
                      opposition = c("Opposition", "reject", "notincorpus"))
dfm(mycorpus, dictionary = mydict2)
```

```
## Not run:
# import the Laver-Garry dictionary from http://bit.ly/1FH2nvf
lgdict <- dictionary(file = "http://www.kenbenoit.net/courses/essex2014qta/LaverGarry.cat",
                     format = "wordstat")
head(dfm(data_corpus_inaugural, dictionary = lgdict))

# import a LIWC formatted dictionary from http://www.moralfoundations.org
mfdict <- dictionary(file = "http://ow.ly/VMRkL", format = "LIWC")
head(dfm(data_corpus_inaugural, dictionary = mfdict))
## End(Not run)
```

`docnames` *get or set document names*

Description

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

Usage

```
docnames(x)

docnames(x) <- 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 a corpus. (Does not work for dfm objects, whose document names are fixed.)

See Also

`featnames`

Examples

```
# query the document names of a corpus
docnames(data_corpus_irishbudget2010)

# query the document names of a tokens object
docnames(tokens(data_char_ukimmig2010))

# query the document names of a dfm
```

```
docnames(dfm(data_corpus_inaugural[1:5]))

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

`docvars` *get or set for document-level variables*

Description

Get or set variables associated with a document in a corpus, or get these variables from a tokens or dfm object.

Usage

```
docvars(x, field = NULL)

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

Arguments

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

Value

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

`docvars<-` assigns `value` to the named `field`

Note

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

Assigning docvars to a tokens object is not supported. (You should only be manipulating these variables at the corpus level.)

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 postive integer value for window. Note: if x is a dfm object, then context can only be "windows".
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 customer 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 seperately. 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)
...	not used here

Details

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

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

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

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

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

Author(s)

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

References

Momtazi, S., Khudanpur, S., & Klakow, D. (2010). "A comparative study of word co-occurrence for term clustering in language model-based sentence retrieval." *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the ACL*, Los Angeles, California, June 2010, pp. 325-328.

Daniel Jurafsky & James H. Martin. (2015) *Speech and Language Processing*. Draft of April 11, 2016. Chapter 16, Semantics with Dense Vectors.

Church, K. W. & P. Hanks (1990) "Word association norms, mutual information, and lexicography" *Computational Linguistics*, 16(1):22–29.

Examples

```
# see http://bit.ly/29b2zOA
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,
```

```
              weights = c(3, 2, 1), ordered = TRUE, tri = FALSE)

# with multiple documents
txts <- c("a a a b b c", "a 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 a dfm in alphabetical order of the features.

Usage

```
fcm_sort(x)
```

Arguments

x	fcm object

Value

A fcm object whose features have been alphabetically sorted

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)
rownames(myfcm)[3] <- colnames(myfcm)[3] <- "Z"
```

```
myfcm
fcm_sort(myfcm)
```

`featnames` *get the feature labels from a dfm*

Description

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

Usage

```
featnames(x)
```

Arguments

`x`	the dfm whose features will be extracted

Value

character vector of the features

Examples

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

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

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

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

`head.dfm` *return the first or last part of a dfm*

Description

For a dfm object, returns the first or last `n` documents and first `ncol` features for inspection.

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 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.
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

```
myDfm <- dfm(data_corpus_inaugural, ngrams = 2, verbose = FALSE)
head(myDfm)
tail(myDfm)
tail(myDfm, 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

See Also

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

---

is.dictionary *check if an object is a dictionary*

Description

Return `TRUE` if an object is a **quanteda** dictionary.

Usage

```
is.dictionary(x)
```

Arguments

x	any object

---

kwic *locate keywords-in-context*

Description

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

Usage

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

is.kwic(x)

## S3 method for class 'kwic'
as.tokens(x)
```

Arguments

x	a character, corpus, or tokens object
keywords	a keyword pattern or phrase consisting of multiple keyword patterns, possibly including punctuation. If a phrase, keywords will be tokenized using the ... options.
window	the number of context words to be displayed around the keyword.
valuetype	how to interpret keyword expressions: "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
...	additional arguments passed to tokens, for applicable object types

Value

A kwic object classed data.frame, with the document name (docname), the token index position (position), the context before (contextPre), the keyword in its original format (keyword, preserving case and attached punctuation), and the context after (contextPost).

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, "war against")
kwic(data_corpus_inaugural, "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 list of the metadata fields in the corpus. If a list is not what you wanted, you can wrap the results in unlist, but this will remove any metadata field that is set to NULL.

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")
```

---

metadoc *get or set document-level meta-data*

Description

Get or set the document-level meta-data.

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.

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 *count the number of documents or features*

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))
```

---

nscrabble *count the Scrabble letter values of text*

Description

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

Usage

```
nscrabble(x, FUN = sum)
```

Arguments

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

Value

a (named) integer vector of 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.

Author(s)

Kenneth Benoit

Examples

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

nsentence *count the number of sentences*

Description

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

Usage

```
nsentence(x, ...)
```

Arguments

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

Value

count(s) of the total sentences per text

Note

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

Examples

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

nsyllable *count syllables in a text*

Description

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

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

Usage

```
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 *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 be different for dfm objects and the texts from which the dfm was generated. Because the method tokenizes the text in order to count the tokens, your results will depend on the options passed through to tokens

Examples

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

# with some real texts
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)))
```

---

quanteda_options *get or set package options for quanteda*

Description

Get or set global options affecting functions across **quatneda**.

Usage

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

Arguments

...	options to be set, as key-value pair, same as options
reset	logical; if TRUE, reset all **quanteda** options to their default values

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 uee parallelization

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

```
quanteda_options()
quanteda_options(verbose = FALSE)
quanteda_options("verbose" = FALSE)
quanteda_options("threads")
## Not run:
quanteda_options(reset = TRUE)

## 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)
```

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))
```

---

stopwords *access built-in stopwords*

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 = "english")
```

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 English stopwords from the SMART information retrieval system (obtained from http://jmlr.csail.mit.edu/papers/volume5/lewis04a/a11-smart-stop-list/english.stop) and a set of stopword lists from the Snowball stemmer project in different languages (see http://snowballstem.org/projects.html). Supported languages are Arabic, Danish, Dutch, English, Finnish, French, German, Greek, Hungarian, Italian, Norwegian, Portuguese, Russian, Spanish, and Swedish. Language names should be lower-case (except for "SMART" – see below) and are case sensitive.

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 encourged to inspect the list and to make sure it fits your particular requirements. The built-in English stopword list does not contain "will", for instance, because of its multiple meanings, but you might want to include this word for your own application.

Examples

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

# adding to the built-in stopword list
toks <- tokenize("The judge will sentence Mr. Adams to nine years in prison", remove_punct = TRUE)
removeFeatures(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 in the **ca** package.

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
`threads`	specifies the number of threads to be used; set to 1 to use a serial version of the function. Only applies 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 sacrify the accuracy. Only applies 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 multiple threads computation, which retains sparsity of all involved matrices. It might not help the speed unless you have a very big dfm.

Author(s)

Kenneth Benoit and Haiyan Wang

References

Nenadic, O. and Greenacre, M. (2007). Correspondence analysis in R, with two- and three-dimensional graphics: The ca package. *Journal of Statistical Software*, 20 (3), http://www.jstatsoft.org/v20/i03/

Examples

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

---

textmodel_NB *Naive Bayes classifier for texts*

Description

Currently working for vectors of texts – not specially defined for a dfm.

Usage

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

Arguments

x	the dfm on which the model will be fit. Does not need to contain only the training documents.
y	vector of training labels associated with each document identified in train. (These will be converted to factors if not already factors.)
smooth	smoothing parameter for feature counts by class
prior	prior distribution on texts, see details
distribution	count model for text features, can be multinomial or Bernoulli
...	more arguments passed through

Details

This naive Bayes model works on word counts, with smoothing.

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

Examples

```
## Example from 13.1 of _An Introduction to Information Retrieval_
trainingset <- as.dfm(matrix(c(1, 2, 0, 0, 0, 0,
                        0, 2, 0, 0, 1, 0,
                        0, 1, 0, 1, 0, 0,
                        0, 1, 1, 0, 0, 1,
                        0, 3, 1, 0, 0, 1),
                      ncol=6, nrow=5, byrow=TRUE,
                      dimnames = list(docs = paste("d", 1:5, sep = ""),
                                      features = c("Beijing", "Chinese",  "Japan", "Macao",
                                                   "Shanghai", "Tokyo"))))
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))
predict(nb.p261, newdata = trainingset[5, ])

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

`textmodel_wordfish` *wordfish text model*

Description

Estimate Slapin and Proksch's (2008) "wordfish" Poisson scaling model of one-dimensional document positions using conditional maximum likelihood.

Usage

```
textmodel_wordfish(x, dir = c(1, 2), priors = c(Inf, Inf, 3, 1),
  tol = c(1e-06, 1e-08), dispersion = c("poisson", "quasipoisson"),
  dispersion_level = c("feature", "overall"), dispersion_floor = 0,
  sparse = 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 α_i, ψ_j, β_j, and θ_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 $\text{se}(\sigma) = 3$, through the third element in `priors`.

Value

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

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

Author(s)

Benjamin Lauderdale, Haiyan Wang, and Kenneth Benoit

References

Jonathan Slapin and Sven-Oliver Proksch. 2008. "A Scaling Model for Estimating Time-Series Party Positions from Texts." *American Journal of Political Science* 52(3):705-772.

Lowe, Will and Kenneth Benoit. 2013. "Validating Estimates of Latent Traits from Textual Data Using Human Judgment as a Benchmark." *Political Analysis* 21(3), 298-313. http://doi.org/10.1093/pan/mpt002

Examples

```
textmodel_wordfish(data_dfm_LBGexample, dir = c(1,5))

## Not run:
ie2010dfm <- dfm(data_corpus_irishbudget2010, verbose = FALSE)
(wfm1 <- textmodel_wordfish(ie2010dfm, dir = c(6,5)))
(wfm2a <- textmodel_wordfish(ie2010dfm, dir = c(6,5),
                             dispersion = "quasipoisson", dispersion_floor = 0))
(wfm2b <- textmodel_wordfish(ie2010dfm, 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(featnames(ie2010dfm) %in% names(topfeatures(ie2010dfm, 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)) {
    wfmodelAustin <- austin::wordfish(quanteda::as.wfm(ie2010dfm), dir = c(6,5))
    cor(wfm1@theta, wfmodelAustin$theta)
}
## End(Not run)
```

`textmodel_wordscores` *Wordscores text model*

Description

`textmodel_wordscores` implements Laver, Benoit and Garry's (2003) wordscores method for scaling of a single dimension.

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

Laver, Michael, Kenneth R Benoit, and John Garry. 2003. "Extracting Policy Positions From Political Texts Using Words as Data." American Political Science Review 97(02): 311-31

Beauchamp, N. 2012. "Using Text to Scale Legislatures with Uninformative Voting." New York University Mimeo.

Martin, L W, and G Vanberg. 2007. "A Robust Transformation Procedure for Interpreting Political Text." Political Analysis 16(1): 93-100.

See Also

`predict.textmodel_wordscores_fitted`

Examples

```
(ws <- textmodel_wordscores(data_dfm_LBGexample, c(seq(-1.5, 1.5, .75), NA)))

predict(ws)
predict(ws, rescaling = "mv")
predict(ws, rescaling = "lbg")

# same as:
(ws2 <- textmodel_wordscores(data_dfm_LBGexample, c(seq(-1.5, 1.5, .75), NA)))
predict(ws2)
```

`textmodel_wordshoal` *wordshoal text model*

Description

Estimate Lauderdale and Herzog's (2016) model for one-dimensional document author (eg speakers) positions based on multiple groups of texts (eg 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	tolerance for convergence. 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

Benjamin E. Lauderdale and Alexander Herzog. (forthcoming) "Measuring Political Positions from Legislative Speech." *Political Analysis*.

Examples

```
## 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(ie30corpus, "debateID"),
                        authors = docvars(ie30corpus, "member.name"))
fitdf <- merge(as.data.frame(summary(wordshoalfit)),
               docvars(ie30corpus),
               by.x="row.names", by.y="member.name")
fitdf <- subset(fitdf,!duplicated(memberID))
aggregate(theta ~ party.name, data = fitdf, mean)

## 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	optional grouping variable for sorting categories of the documents. Only applies when margin = "documents".

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

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
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
textplot_scale1d(wfm, margin = "features",
                 highlighted = c("government", "global", "children",
                                 "bank", "economy", "the", "citizenship",
                                 "productivity", "deficit"))
## 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

```
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 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 word cloud 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 ggplot object, you can modify the plot by adding ggplot 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

`plot.kwic` returns a ggplot object

Author(s)

Adam Obeng

Examples

```
## 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)
```

`texts` *get or assign corpus texts*

Description

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

Usage

```
texts(x, groups = NULL, spacer = "  ")

texts(x) <- value

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

Arguments

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

Details

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

Value

For `texts`, a character vector of the texts in the corpus.

For `texts <-`, the corpus with the updated texts.

for `texts <-`, a corpus with the texts replaced by `value`

`as.character(x)` is equivalent to `texts(x)`

Note

You are strongly encouraged as a good practice of text analysis workflow *not* to modify the substance of the texts in a corpus. Rather, this sort of processing is better performed through downstream operations. For instance, do not lowercase the texts in a corpus, or you will never be able to recover the original case. Rather, apply `tokens_tolower` after applying `tokens` to a corpus, or use the option `tolower = TRUE` in `dfm`..

Examples

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

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

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

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

`textstat_collocations` *calculate collocation statistics*

Description

Identify and score collocations from a corpus, character, or tokens object, with targeted selection.

#' @rdname textstat_collocations #' @noRd #' @export textstat_collocations.character <- function(x, method = c("lr", "chi2", "pmi", "dice", "bj"), ...) method <- match.arg(method) textstat_collocations(tokens(x), method = method, ...)

Usage

```
textstat_collocations(x, method = c("lr", "chi2", "pmi", "dice", "bj"),
  max_size = 3, min_count = 2, ...)

is.collocations(x)
```

Arguments

x	a character, corpus, or tokens object to be mined for collocations
method	association measure for detecting collocations. Let i index documents, and j index features, n_{ij} refers to observed counts, and m_{ij} the expected counts in a collocations frequency table of dimensions $(J - size + 1)^2$. Available measures are computed as:
	"lr" The likelihood ratio statistic G^2, computed as: $$2 * \sum_i \sum_j (n_{ij} * log \frac{n_{ij}}{m_{ij}})$$
	"chi2" Pearson's χ^2 statistic, computed as: $$\sum_i \sum_j \frac{(n_{ij} - m_{ij})^2}{m_{ij}}$$
	"pmi" point-wise mutual information score, computed as log n_{11}/m_{11}
	"dice" the Dice coefficient, computed as $n_{11}/n_{1\cdot} + n_{\cdot 1}$
	"bj" Blaheta and Johnson's method (called through sequences)
max_size	numeric argument representing the maximum length of the collocations to be scored. The maximum size is currently 3 for all methods except "bj", which has a maximum size of 5.
min_count	minimum frequency of collocations that will be scored
...	additional arguments passed to collocations2 for the first four methods, or to sequences for method = "bj"

Details

#' @rdname textstat_collocations #' @noRd #' @export textstat_collocations.tokenizedTexts <- function(x, method = c("lr", "chi2", "pmi", "dice", "bj"), ...) method <- match.arg(method) textstat_collocations(as.tokens(x), method = method, ...)

check if an object is collocations object

Value

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

Note

This function is under active development, and we aim to improve both its operation and efficiency in the next release of **quanteda**.

References

Blaheta, D., & Johnson, M. (2001). Unsupervised learning of multi-word verbs. Presented at the ACLEACL Workshop on the Computational Extraction, Analysis and Exploitation of Collocations.

McInnes, B T. 2004. "Extending the Log Likelihood Measure to Improve Collocation Identification." M.Sc. Thesis, University of Minnesota.

Examples

```
txts <- c("quanteda is a package for quantitative text analysis",
          "quantitative text analysis is a rapidly growing field",
          "The population is rapidly growing")
toks <- tokens(txts)
textstat_collocations(toks, method = "lr")
textstat_collocations(toks, method = "lr", min_count = 1)
textstat_collocations(toks, method = "lr", max_size = 3, min_count = 1)
(cols <- textstat_collocations(toks, method = "lr", max_size = 3, min_count = 2))
as.tokens(cols)

# extracting multi-part proper nouns (capitalized terms)
toks2 <- tokens(corpus_segment(data_corpus_inaugural, what = "sentence"))
toks2 <- tokens_select(toks2, stopwords("english"), "remove", padding = TRUE)
seqs <- textstat_collocations(toks2, method = "bj",
                              features = "^([A-Z][a-z\\-]{2,})",
                              valuetype = "regex", case_insensitive = FALSE)
head(seqs, 10)

# compounding tokens is more efficient when applied to the same tokens object
toks_comp <- tokens_compound(toks2, seqs)
```

---

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).

Usage

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

textstat_simil(x, selection = NULL, n = 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.
n	the top n highest-ranking items will be returned. If n is NULL, return all items. Useful if the output object will be coerced into a list, for instance if the top n most similar features to a target feature is desired. (See examples.)
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".

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 weight(x, "relFreq").

Author(s)

Kenneth Benoit, Haiyan Wang

See Also

textstat_dist, as.list.dist, dist

Examples

```
# 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(d1)

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

# similarities for 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", n = 8))
as.list(s2)
```

`textstat_keyness` *calculate keyness statistics*

Description

calculate keyness statistics

Usage

```
textstat_keyness(x, target = 1L, measure = c("chi2", "exact", "lr"),
  sort = TRUE)
```

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. Currenly available: `"chi2"` (chi^2 with Yates correction); `"exact"` (Fisher's exact test); `"lr"` for the likelihood ratio G statistic with Yates correction.
`sort`	logical; if `TRUE` sort features scored in descending order of the measure, otherwise leave in original feature order

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 statistic.

References

Bondi, Marina, and Mike Scott, eds. 2010. *Keyness in Texts*. Amsterdam, Philadelphia: John Benjamins, 2010.

Stubbs, Michael. 2010. "Three Concepts of Keywords". In *Keyness in Texts*, Marina Bondi and Mike Scott, eds. pp21–42. Amsterdam, Philadelphia: John Benjamins.

Scott, M. & Tribble, C. 2006. *Textual Patterns: keyword and corpus analysis in language education*. Amsterdam: Benjamins, p. 55.

Dunning, Ted. 1993. "Accurate Methods for the Statistics of Surprise and Coincidence", *Computational Linguistics*, Vol 19, No. 1, pp. 61-74.

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
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 preseidents
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

```
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:

$$K = 10^4 \times \frac{(\sum_{X=1}^{X} f_X X^2) - N}{N^2}$$

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 **koRpus** package.

References

Covington, M.A. & McFall, J.D. (2010). Cutting the Gordian Knot: The Moving-Average Type-Token Ratio (MATTR). *Journal of Quantitative Linguistics*, 17(2), 94–100.

Maas, H.-D., (1972). \"Uber den Zusammenhang zwischen Wortschatzumfang und L\"ange eines Textes. *Zeitschrift f\"ur Literaturwissenschaft und Linguistik*, 2(8), 73–96.

McCarthy, P.M. & Jarvis, S. (2007). vocd: A theoretical and empirical evaluation. *Language Testing*, 24(4), 459–488.

McCarthy, P.M. & Jarvis, S. (2010). MTLD, vocd-D, and HD-D: A validation study of sophisticated approaces to lexical diversity assessment. *Behaviour Research Methods*, 42(2), 381–392.

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

Tweedie. F.J. & Baayen, R.H. (1998). How Variable May a Constant Be? Measures of Lexical Richness in Perspective. *Computers and the Humanities*, 32(5), 323–352.

Examples

```
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)
```

---

textstat_readability *calculate readability*

Description

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

Usage

```
textstat_readability(x, measure = c("all", "ARI", "ARI.simple", "Bormuth",
  "Bormuth.GP", "Coleman", "Coleman.C2", "Coleman.Liau", "Coleman.Liau.grade",
  "Coleman.Liau.short", "Dale.Chall", "Dale.Chall.old", "Dale.Chall.PSK",
  "Danielson.Bryan", "Danielson.Bryan.2", "Dickes.Steiwer", "DRP", "ELF",
  "Farr.Jenkins.Paterson", "Flesch", "Flesch.PSK", "Flesch.Kincaid", "FOG",
  "FOG.PSK", "FOG.NRI", "FORCAST", "FORCAST.RGL", "Fucks", "Linsear.Write",
  "LIW", "nWS", "nWS.2", "nWS.3", "nWS.4", "RIX", "Scrabble", "SMOG", "SMOG.C",
  "SMOG.simple",      "SMOG.de", "Spache", "Spache.old", "Strain",
  "Traenkle.Bailer", "Traenkle.Bailer.2", "Wheeler.Smith", "meanSentenceLength",
  "meanWordSyllables"), remove_hyphens = TRUE, min_sentence_length = 1,
  max_sentence_length = 10000, drop = TRUE, ...)
```

Arguments

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

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)
```

---

tokens *tokenize a set of texts*

Description

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

Usage

```
tokens(x, what = c("word", "sentence", "character", "fastestword",
  "fasterword"), remove_numbers = FALSE, remove_punct = FALSE,
  remove_symbols = 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 or corpus 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{
	"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.
...	additional arguments not used

Details

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

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`

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!
txt <- "Repo https://githib.com/kbenoit/quanteda, and www.stackoverflow.com."
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\t4ever, 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")
tokens("Great website: http://textasdata.com?page=123.", what = "character",
         remove_separators = FALSE)

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

# removing features (stopwords) from tokenized texts
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, sequences, concatenator = "_", valuetype = c("glob",
  "regex", "fixed"), case_insensitive = TRUE, join = FALSE)
```

Arguments

x	an input tokens object
sequences	the input sequence, one of: • character vector, whose elements will be split on whitespace; • list of characters, consisting of a list of token patterns, separated by white space; • tokens object; • dictionary object; • collocations object.
concatenator	the concatenation character that will connect the words making up the multi-word sequences. The default _ is highly 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	how to interpret keyword expressions: "glob" for "glob"-style wildcard expressions; "regex" for regular expressions; or "fixed" for exact matching. See valuetype for details.
case_insensitive	logical; if TRUE, ignore case when matching
join	logical; if TRUE, join overlapped compounds

Value

a tokens object in which the token sequences matching the patterns in sequences have been replaced by compound "tokens" joined by the concatenator

Author(s)

Kenneth Benoit (R) and Kohei Watanabe (C++)

Examples

```
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("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"), concatenator = " ", case_insensitive = TRUE,
  capkeys = !exclusive, exclusive = TRUE, multiword = TRUE,
  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`	how to interpret keyword expressions: `"glob"` for "glob"-style wildcard expressions; `"regex"` for regular expressions; or `"fixed"` for exact matching. See valuetype for details.
`concatenator`	a charactor that connect words in multi-words entries in `x`
`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

multiword	if FALSE, multi-word entries in dictionary are treated as single 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))

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"),
                                  oceans = c("Atlantic", "Pacific")),
                        Europe = list(Countries = c("Britain", "Ireland"),
                                      oceans = list(west = "Irish Sea",
                                                    east = "English Channel"))))
tokens_lookup(toks, dict, levels = 1)
tokens_lookup(toks, dict, levels = 2)
tokens_lookup(toks, dict, levels = 1:2)
tokens_lookup(toks, dict, levels = 3)
tokens_lookup(toks, dict, levels = c(1,3))
tokens_lookup(toks, dict, levels = c(2,3))
```

---

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 = "_")

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

Guthrie, D., B. Allison, W. Liu, and L. Guthrie. 2006. "A Closer Look at Skip-Gram Modelling."

Examples

```
# ngrams
tokens_ngrams(tokens(c("a b c d e", "c d e f g")), n = 2:3)

toks <- tokens(c(text1 = "the quick brown fox jumped over the lazy dog"))
tokens_ngrams(toks, n = 1:3)
```

```
tokens_ngrams(toks, n = c(2,4), concatenator = " ")
tokens_ngrams(toks, n = c(2,4), skip = 1, concatenator = " ")

# on character
char_ngrams(letters[1:3], n = 1:3)

# skipgrams
toks <- tokens("insurgents killed in ongoing fighting")
tokens_skipgrams(toks, n = 2, skip = 0:1, concatenator = " ")
tokens_skipgrams(toks, n = 2, skip = 0:2, concatenator = " ")
tokens_skipgrams(toks, n = 3, skip = 0:2, concatenator = " ")
```

`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, features)` defined as a shortcut for `tokens_select(x, features, 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 only positive features from a list of regular expressions, including a dictionary.

Usage

```
tokens_select(x, features, selection = c("keep", "remove"),
  valuetype = c("glob", "regex", "fixed"), case_insensitive = TRUE,
  padding = FALSE, verbose = quanteda_options("verbose"))

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

Arguments

`x`	tokens object whose token elements will be selected
`features`	one of: a character vector of features to be selected, a dictionary class object whose values (not keys) will provide the features to be selected.
`selection`	whether to `"keep"` or `"remove"` the features
`valuetype`	how to interpret keyword expressions: `"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`	(only for `tokenizedTexts` objects) 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 features, for instance if a window of adjacency needs to be computed.

verbose if TRUE print messages about how many features were removed

Value

a tokens object with features removed

Examples

```
## for tokenized texts
txt <- c(wash1 <- "Fellow citizens, I am again called upon by the voice of my country to
                   execute the functions of its Chief Magistrate.",
         wash2 <- "When the occasion proper for it shall arrive, I shall endeavor to express
                   the high sense I entertain of this distinguished honor.")
tokens_remove(tokens(txt, remove_punct = TRUE), stopwords("english"))
```

---

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, ...)

tokens_toupper(x, ...)
```

Arguments

x	a tokens object
...	additional arguments passed to char_tolower (currently this is limited to keep_acronyms)

See Also

char_tolower, char_toupper

Examples

```
# for a document-feature matrix
toks <- tokens(c(txt1 = "b A A", txt2 = "C C a b B"))
tokens_tolower(toks)
tokens_toupper(toks)
```

tokens_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

```
tokens_wordstem(x, language = "porter")

char_wordstem(x, language = "porter")

dfm_wordstem(x, language = "porter")
```

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://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` *list the most frequent features*

Description

List the most (or least) frequently occuring features in a dfm.

Usage

```
topfeatures(x, n = 10, decreasing = TRUE, ci = 0.95)
```

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, if FALSE, return the `n` least frequent features
`ci`	confidence interval from 0-1.0 for use if dfm is resampled

Value

A named numeric vector of feature counts, where the names are the feature labels.

Examples

```
# most frequent features
topfeatures(dfm(corpus_subset(data_corpus_inaugural, Year > 1980), verbose = FALSE))
topfeatures(dfm(corpus_subset(data_corpus_inaugural, Year > 1980),
            remove = stopwords("english"), verbose = FALSE))

# least frequent features
topfeatures(dfm(corpus_subset(data_corpus_inaugural, Year > 1980), verbose = FALSE),
            decreasing = FALSE)
```

Index