Package: quanteda (via r-universe)

October 21, 2026

Version 4.5.1

Title Quantitative Analysis of Textual Data

Description A fast, flexible, and comprehensive framework for quantitative text analysis in R. Provides functionality for corpus management, creating and manipulating tokens and n-grams, exploring keywords in context, forming and manipulating sparse matrices of documents by features and feature co-occurrences, analyzing keywords, computing feature similarities and distances, applying content dictionaries, applying supervised and unsupervised machine learning, visually representing text and text analyses, and more.

License GPL-3

Depends R (>= 4.1.0), methods

Imports fastmatch, jsonlite, lifecycle, magrittr, Matrix (>= 1.5-0), Rcpp (>= 0.12.12), SnowballC, stopwords, stringi, xml2, yaml

LinkingTo Rcpp

NeedsCompilation yes

Suggests rmarkdown, spelling, testthat, formatR, tm (>= 0.6), knitr, lsa, rlang, slam, torch

Enhances dplyr, lda, purrr, spacyr, stm, text2vec, tibble, tidytext, tokenizers, topicmodels

URL https://quanteda.io

Encoding UTF-8

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

LazyData TRUE

VignetteBuilder knitr

Language en-GB

Collate 'RcppExports.R' 'tokenizers.R' 'meta.R' 'quanteda-documentation.R' 'aaa.R' 'bootstrap_dfm.R' 'casechange-functions.R' 'char_select.R' 'convert.R'

'corpus-addsummary-metadata.R' 'corpus-methods.R' 'corpus.R'
'corpus_chunk.R' 'corpus_group.R' 'corpus_reshape.R'
'corpus_sample.R' 'corpus_segment.R' 'corpus_subset.R'
'corpus_trim.R' 'data-documentation.R' 'dfm-classes.R'
'dfm-methods.R' 'dfm-print.R' 'dfm-subsetting.R' 'dfm.R'
'dfm_compress.R' 'dfm_group.R' 'dfm_lookup.R' 'dfm_match.R'
'dfm_replace.R' 'dfm_sample.R' 'dfm_select.R' 'dfm_sort.R'
'dfm_subset.R' 'dfm_trim.R' 'dfm_weight.R' 'dictionaries.R'
'dimnames.R' 'fcm-classes.R' 'docnames.R' 'docvars.R'
'fcm-methods.R' 'fcm-print.R' 'fcm-subsetting.R' 'fcm.R'
'fcm_select.R' 'index.R' 'kwic.R' 'message.R' 'nfunctions.R'
'object-builder.R' 'object2fixed.R' 'pattern2fixed.R'
'phrases.R' 'quanteda-package.R' 'quanteda_options.R'
'spacyr-methods.R' 'stopwords.R' 'summary.R' 'textmodel.R'
'textplot.R' 'texts.R' 'textstat.R' 'tokens-methods.R'
'tokens.R' 'tokens_annotate.R' 'tokens_chunk.R'
'tokens_compound.R' 'tokens_group.R' 'tokens_lookup.R'
'tokens_match.R' 'tokens_ngrams.R' 'tokens_recompile.R'
'tokens_replace.R' 'tokens_restore.R' 'tokens_sample.R'
'tokens_segment.R' 'tokens_select.R' 'tokens_split.R'
'tokens_subset.R' 'tokens_trim.R' 'tokens_xptr.R' 'utils.R'
'validator.R' 'wordstem.R' 'zzz.R'

Config/roxygen2/version 8.1.0

Author Kenneth Benoit [cre, aut, cph] (ORCID:
<https://orcid.org/0000-0002-0797-564X>), Kohei Watanabe [aut]
(ORCID: <https://orcid.org/0000-0001-6519-5265>), Haiyan Wang
[aut] (ORCID: <https://orcid.org/0000-0003-4992-4311>), Paul
Nulty [aut] (ORCID: <https://orcid.org/0000-0002-7214-4666>),
Adam Obeng [aut] (ORCID:
<https://orcid.org/0000-0002-2906-4775>), Stefan Müller [aut]
(ORCID: <https://orcid.org/0000-0002-6315-4125>), Akitaka
Matsuo [aut] (ORCID: <https://orcid.org/0000-0002-3323-6330>),
William Lowe [aut] (ORCID:
<https://orcid.org/0000-0002-1549-6163>), Christian Müller
[ctb], Olivier Delmarcelle [ctb] (ORCID:
<https://orcid.org/0000-0003-4347-070X>), European Research
Council [fnd] (ERC-2011-StG 283794-QUANTESS)

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

Repository https://cran.r-universe.dev

Date/Publication 2026-10-21 15:24:41 UTC

RemoteUrl https://github.com/cran/quanteda

RemoteRef HEAD

RemoteSha ffeb7fb5b5e89b0a939422580ccafe96ffb3903b

Contents

as.character.corpus . . . 4
as.dfm . . . 5
as.dictionary . . . 6
as.fcm . . . 7
as.list.tokens . . . 8
as.matrix.dfm . . . 10
as.yaml . . . 10
bootstrap_dfm . . . 11
char_select . . . 12
char_tolower . . . 13
concat . . . 14
convert . . . 15
corpus . . . 17
corpus_chunk . . . 20
corpus_group . . . 21
corpus_reshape . . . 22
corpus_sample . . . 23
corpus_segment . . . 25
corpus_subset . . . 27
corpus_trim . . . 28
data_char_sampletext . . . 30
data_char_ukimmig2010 . . . 30
data_corpus_inaugural . . . 31
data_dfm_lbgexample . . . 32
data_dictionary_LSD2015 . . . 32
dfm . . . 34
dfm_compress . . . 35
dfm_group . . . 37
dfm_lookup . . . 38
dfm_match . . . 40
dfm_replace . . . 41
dfm_sample . . . 42
dfm_select . . . 43
dfm_sort . . . 46
dfm_subset . . . 46
dfm_tfidf . . . 48
dfm_tolower . . . 49
dfm_trim . . . 50
dfm_weight . . . 52
dictionary . . . 54
docfreq . . . 57
docnames . . . 59
docvars . . . 60
fcm . . . 62
fcm_sort . . . 64
featfreq . . . 65

featnames . . . 66
index . . . 66
is.collocations . . . 67
kwic . . . 68
meta . . . 69
ndoc . . . 70
nsentence . . . 71
ntoken . . . 72
phrase . . . 73
print-methods . . . 74
quanteda_options . . . 76
spacyr-methods . . . 78
sparsity . . . 79
textmodels . . . 79
textplots . . . 79
textstats . . . 80
tokens . . . 80
tokens_annotate . . . 84
tokens_chunk . . . 85
tokens_compound . . . 86
tokens_group . . . 88
tokens_lookup . . . 89
tokens_match . . . 92
tokens_ngrams . . . 93
tokens_replace . . . 95
tokens_sample . . . 96
tokens_segment . . . 98
tokens_select . . . 99
tokens_split . . . 101
tokens_subset . . . 102
tokens_tolower . . . 104
tokens_trim . . . 104
tokens_wordstem . . . 106
tokens_xptr . . . 108
topfeatures . . . 108
types . . . 110

Index **111**

`as.character.corpus` *Coercion and checking methods for corpus objects*

Description

Coercion functions to and from corpus objects, including conversion to a plain character object; and checks for whether an object is a corpus.

Usage

```
## S3 method for class 'corpus'
as.character(x, use.names = TRUE, ...)

is.corpus(x)

as.corpus(x)
```

Arguments

x	object to be coerced or checked
use.names	logical; preserve (document) names if TRUE
...	additional arguments used by specific methods

Value

`as.character()` returns the corpus as a plain character vector, with or without named elements.

`is.corpus` returns TRUE if the object is a corpus.

`as.corpus()` upgrades a corpus object to the newest format. object.

Note

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

---

as.dfm *Coercion and checking functions for dfm objects*

Description

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

Usage

```
as.dfm(x)

is.dfm(x)
```

Arguments

x	a candidate object for checking or coercion to dfm

Value

`as.dfm` converts an input object into a dfm. Row names are used for docnames, and column names for featnames, of the resulting dfm.

`is.dfm` returns `TRUE` if and only if its argument is a dfm.

See Also

`as.data.frame.dfm()`, `as.matrix.dfm()`, `convert()`

---

as.dictionary *Coercion and checking functions for dictionary objects*

Description

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

Usage

```
as.dictionary(x, ...)

## S3 method for class 'data.frame'
as.dictionary(x, format = c("tidytext"), separator = " ", tolower = FALSE, ...)

is.dictionary(x)
```

Arguments

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

Value

`as.dictionary` returns a **quanteda** dictionary object. This conversion function differs from the `dictionary()` constructor function in that it converts an existing object rather than creates one from components or from a file.

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

Examples

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

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

## End(Not run)

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

`as.fcm` *Coercion and checking functions for fcm objects*

Description

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

Usage

```
as.fcm(x)
```

Arguments

x	a candidate object for checking or coercion to dfm

Value

`as.fcm` converts an input object into a fcm.

`as.list.tokens` *Coercion, checking, and combining functions for tokens objects*

Description

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

Usage

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

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

is.tokens(x)

as.tensor(x, ...)

## S3 method for class 'tokens'
as.tensor(x, length = NULL, extract = NULL, ...)

## S3 method for class 'tokens'
as.matrix(x, length = NULL, extract = NULL, drop = TRUE, ...)

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

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

is.tokens(x)
```

Arguments

`x`	object to be coerced or checked
`...`	additional arguments used by specific methods. For c.tokens, these are the tokens objects to be concatenated.
`use.names`	logical; preserve names if `TRUE`. For `as.character` and `unlist` only.

length	optional integer specifying the maximum length (number of positions) for the tensor. If NULL (default), the length is inferred from the maximum token position across all documents.
extract	indices for documents to be extracted from x.
drop	if TRUE returns a vector instead of a matrix with only one row.
concatenator	character; the concatenation character that will connect the tokens making up a multi-token sequence. See details.
include_pos	character; whether and which part-of-speech tag to use: "none" do not use any part of speech indicator, "pos" use the pos variable, "tag" use the tag variable. The POS will be added to the token after "concatenator".
use_lemma	logical; if TRUE, use the lemma rather than the raw token

Details

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

Value

as.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.

as.tensor returns a tensor from a tokens object, compatible with the **torch** package. Each document is represented as a row, and token positions as columns. The integer token IDs in the resulting tensor are shifted by one to comply with 1-based indexing in R.

as.tokens returns a quanteda tokens object.

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

Examples

```
## Not run:
library(torch)
toks <- tokens(c(doc1 = "a b c d e f g",
                 doc2 = "a b c g",
                 doc3 = ""))
as.tensor(toks)

## End(Not run)

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

```
              "and", "the", "Pacific-Ocean", "."))
toks <- as.tokens(lis, concatenator = "-")
tokens_lookup(toks, dict)
```

`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, ...)
```

Arguments

`x`	dfm to be coerced
`...`	unused

Examples

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

`as.yaml` *Convert quanteda dictionary objects to the YAML format*

Description

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

Usage

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

Arguments

`x`	a dictionary object

Value

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

Examples

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

## End(Not run)
```

---

bootstrap_dfm *Bootstrap a dfm*

Description

Create an array of resampled dfms.

Usage

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

Arguments

x	a dfm object
n	number of resamples
...	additional arguments passed to dfm()
verbose	if TRUE print status messages

Details

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

Value

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

Author(s)

Kenneth Benoit

Examples

```
set.seed(10)
txt <- c(textone = "This is a sentence.  Another sentence.  Yet another.",
         texttwo = "Premiere phrase.  Deuxieme phrase.")
dfmat <- corpus_reshape(corpus(txt), to = "sentences") |>
    tokens() |>
    dfm()
bootstrap_dfm(dfmat, n = 3)
```

`char_select` *Select or remove elements from a character vector*

Description

These function select or discard elements from a character object. For convenience, the functions `char_remove` and `char_keep` are defined as shortcuts for `char_select(x, pattern, selection = "remove")` and `char_select(x, pattern, selection = "keep")`, respectively.

These functions make it easy to change, for instance, stopwords based on pattern matching.

Usage

```
char_select(
  x,
  pattern,
  selection = c("keep", "remove"),
  valuetype = c("glob", "fixed", "regex"),
  case_insensitive = TRUE
)

char_remove(x, ...)

char_keep(x, ...)
```

Arguments

`x`	an input character vector
`pattern`	a character vector, list of character vectors, dictionary, or collocations object. See pattern for details.
`selection`	whether to `"keep"` or `"remove"` the tokens matching `pattern`
`valuetype`	the type of pattern matching: `"glob"` for "glob"-style wildcard expressions; `"regex"` for regular expressions; or `"fixed"` for exact matching. See valuetype for details.
`case_insensitive`	logical; if `TRUE`, ignore case when matching a `pattern` or dictionary values
`...`	additional arguments passed by `char_remove` and `char_keep` to `char_select`. Cannot include `selection`.

Value

a modified character vector

Examples

```
# character selection
mykeywords <- c("natural", "national", "denatured", "other")
char_select(mykeywords, "nat*", valuetype = "glob")
char_select(mykeywords, "nat", valuetype = "regex")
char_select(mykeywords, c("natur*", "other"))
char_select(mykeywords, c("natur*", "other"), selection = "remove")

# character removal
char_remove(letters[1:5], c("a", "c", "x"))
words <- c("any", "and", "Anna", "as", "announce", "but")
char_remove(words, "an*")
char_remove(words, "an*", case_insensitive = FALSE)
char_remove(words, "^.n.+$", valuetype = "regex")

# remove some of the system stopwords
stopwords("en", source = "snowball")[1:6]
stopwords("en", source = "snowball")[1:6] |>
  char_remove(c("me", "my*"))

# character keep
char_keep(letters[1:5], c("a", "c", "x"))
```

---

char_tolower *Convert the case of character objects*

Description

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

Usage

```
char_tolower(x, keep_acronyms = FALSE)

char_toupper(x)
```

Arguments

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

Examples

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

# 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)
```

`concat` *Return the concatenator character from an object*

Description

Get the concatenator character from a tokens object.

Usage

```
concat(x)

concatenator(x)
```

Arguments

x	a tokens object

Details

The concatenator character is a special delimiter used to link separate tokens in multi-token phrases. It is embedded in the meta-data of tokens objects and used in downstream operations, such as `tokens_compound()` or `tokens_lookup()`. It can be extracted using `concat()` and set using `tokens(x, concatenator = ...)` when x is a tokens object.

The default _ is recommended since it will not be removed during normal cleaning and tokenization (while nearly all other punctuation characters, at least those in the Unicode punctuation class `[P]` will be removed).

Value

a character of length 1

Examples

```
toks <- tokens(data_corpus_inaugural[1:5])
concat(toks)
```

convert *Convert quanteda objects to non-quanteda formats*

Description

Convert a quanteda dfm or corpus object to a format useable by other 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. For corpus objects, `convert` provides an easy way to make a corpus and its document variables into a data.frame.

Usage

```
convert(x, to, ...)

## S3 method for class 'dfm'
convert(
  x,
 to = c("lda", "tm", "stm", "austin", "topicmodels", "lsa", "matrix", "data.frame",
    "tripletlist"),
  docvars = NULL,
  omit_empty = TRUE,
  docid_field = "doc_id",
  ...
)

## S3 method for class 'corpus'
convert(x, to = c("data.frame", "json"), pretty = FALSE, ...)
```

Arguments

x	a dfm or corpus to be converted
to	target conversion format, one of:
	`"lda"` a list with components "documents" and "vocab" as needed by the function lda.collapsed.gibbs.sampler from the **lda** package
	`"tm"` a DocumentTermMatrix from the **tm** package. Note: The **tm** package version of `as.TermDocumentMatrix()` allows a `weighting` argument, which supplies a weighting function for `TermDocumentMatrix()`. Here the default is for term frequency weighting. If you want a different weighting, apply the weights after converting using one of the **tm** functions. For other available weighting functions from the **tm** package, see TermDocumentMatrix.
	`"stm"` the format for the **stm** package
	`"austin"` the `wfm` format from the **austin** package
	`"topicmodels"` the "dtm" format as used by the **topicmodels** package
	`"lsa"` the "textmatrix" format as used by the **lsa** package

	`"data.frame"` a data.frame of without row.names, in which documents are rows, and each feature is a variable (for a dfm), or each text and its document variables form a row (for a corpus)
	`"json"` (corpus only) convert a corpus and its document variables into JSON format, using the format described in jsonlite::toJSON()
	`"tripletlist"` a named "triplet" format list consisting of `document`, `feature`, and `frequency`
`...`	unused directly
`docvars`	optional data.frame of document variables used as the `meta` information in conversion to the **stm** package format. This aids in selecting the document variables only corresponding to the documents with non-zero counts. Only affects the "stm" format.
`omit_empty`	logical; if `TRUE`, omit empty documents and features from the converted dfm. This is required for some formats (such as STM) that do not accept empty documents. Only used when `to = "lda"` or `to = "topicmodels"`. For `to = "stm"` format, `omit_empty` is always `TRUE`.
`docid_field`	character; the name of the column containing document names used when `to = "data.frame"`. Unused for other conversions.
`pretty`	adds indentation whitespace to JSON output. Can be TRUE/FALSE or a number specifying the number of spaces to indent (default is 2). Use a negative number for tabs instead of spaces.

Value

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

Examples

```
## convert a dfm

toks <- corpus_subset(data_corpus_inaugural, Year > 1970) |>
    tokens()
dfmat1 <- dfm(toks)

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

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

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

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

```
str(tmdfm)

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

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

## End(Not run)

## convert a corpus into a data.frame

corp <- corpus(c(d1 = "Text one.", d2 = "Text two."),
               docvars = data.frame(dvar1 = 1:2, dvar2 = c("one", "two"),
                                    stringsAsFactors = FALSE))
convert(corp, to = "data.frame")
convert(corp, to = "json")
```

`corpus` *Construct a corpus object*

Description

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

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

Usage

```
corpus(x, ...)

## S3 method for class 'corpus'
corpus(
  x,
  docnames = quanteda::docnames(x),
  docvars = quanteda::docvars(x),
  meta = quanteda::meta(x),
  ...
```

```
)

## S3 method for class 'character'
corpus(
  x,
  docnames = NULL,
  docvars = NULL,
  meta = list(),
  unique_docnames = TRUE,
  ...
)

## S3 method for class 'data.frame'
corpus(
  x,
  docid_field = "doc_id",
  text_field = "text",
  meta = list(),
  unique_docnames = TRUE,
  ...
)

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

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

Arguments

x	a valid corpus source object
...	not used directly
docnames	Names to be assigned to the texts. Defaults to the names of the character vector (if any); doc_id for a data.frame; the document names in a **tm** corpus; or a vector of user-supplied labels equal in length to the number of documents. If none of these are round, then "text1", "text2", etc. are assigned automatically.
docvars	a data.frame of document-level variables associated with each text
meta	a named list that will be added to the corpus as corpus-level, user meta-data. This can later be accessed or updated using meta().

`unique_docnames`
: logical; if `TRUE`, enforce strict uniqueness in `docnames`; otherwise, rename duplicated docnames using an added serial number, and treat them as segments of the same document.

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

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

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

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

`concatenator`
: character between tokens, default is the whitespace.

Details

The texts and document variables of corpus objects can also be accessed using index notation and the `$` operator for accessing or assigning docvars. For details, see `[.corpus()`.

Value

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

For **quanteda** >= 2.0, this is a specially classed character vector. It has many additional attributes but **you should not access these attributes directly**, especially if you are another package author. 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. Using the accessor and replacement functions ensures that future code to manipulate corpus objects will continue to work.

See Also

corpus, `docvars()`, `meta()`, `as.character.corpus()`, `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)

# import a tm VCorpus
```

```
if (requireNamespace("tm", quietly = TRUE)) {
    data(crude, package = "tm")    # load in a tm example VCorpus
    vcorp <- corpus(crude)
    summary(vcorp)

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

    vcorp2 <- tm::VCorpus(tm::VectorSource(data_char_ukimmig2010))
    corp <- corpus(vcorp2)
    summary(corp)
}

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

---

corpus_chunk *Segment a corpus into chunks of a given size*

Description

Segment a corpus into new documents of roughly equal sized text chunks, with the possibility of overlapping the chunks.

Usage

```
corpus_chunk(
  x,
  size,
  truncate = FALSE,
  use_docvars = TRUE,
  verbose = quanteda_options("verbose")
)
```

Arguments

x	tokens object whose token elements will be segmented into chunks
size	integer; the (approximate) token length of the chunks. See Details.
truncate	logical; if TRUE, truncate the text after size

use_docvars	if TRUE, repeat the docvar values for each chunk; if FALSE, drop the docvars in the chunked tokens
verbose	if TRUE print a summary of the object returned by the function: the number of documents and, depending on the object, the number of characters, tokens and types, or features and occurrences. The number of tokens includes paddings, but the number of features excludes them.

Details

The token length is estimated using `stringi::stri_length(txt) / stringi::stri_count_boundaries(txt)` to avoid needing to tokenize and rejoin the corpus from the tokens.

Note that when used for chunking texts prior to sending to large language models (LLMs) with limited input token lengths, size should typically be set to approximately 0.75-0.80 of the LLM's token limit. This is because tokenizers (such as LLaMA's SentencePiece Byte-Pair Encoding tokenizer) require more tokens than the linguistically defined grammatically-based tokenizer that is the **quanteda** default. Note also that because of the use of `stringi::stri_count_boundaries(txt)` to approximate token length (efficiently), the exact token length for chunking will be approximate.

See Also

`tokens_chunk()`

Examples

```
data_corpus_inaugural[1] |>
  corpus_chunk(size = 10)
```

---

corpus_group *Combine documents in corpus by a grouping variable*

Description

Combine documents in a corpus object by a grouping variable, by concatenating their texts in the order of the documents within each grouping variable.

Usage

```
corpus_group(x, groups = docid(x), fill = FALSE, concatenator = " ")
```

Arguments

x	corpus object
groups	grouping variable for sampling, equal in length to the number of documents. This will be evaluated in the docvars data.frame, so that docvars may be referred to by name without quoting. This also changes previous behaviours for groups. See `news(Version >= "3.0", package = "quanteda")` for details.

fill	logical; if TRUE and groups is a factor, then use all levels of the factor when forming the new documents of the grouped object. This will result in a new "document" with empty content for levels not observed, but for which an empty document may be needed. If groups is a factor of dates, for instance, then fill = TRUE ensures that the new object will consist of one new "document" by date, regardless of whether any documents previously existed with that date. Has no effect if the groups variable(s) are not factors.
concatenator	the concatenation character that will connect the grouped documents.

Value

a corpus object whose documents are equal to the unique group combinations, and whose texts are the concatenations of the texts by group. Document-level variables that have no variation within groups are saved in docvars. Document-level variables that are lists are dropped from grouping, even when these exhibit no variation within groups.

Examples

```
corp <- corpus(c("a a b", "a b c c", "a c d d", "a c c d"),
               docvars = data.frame(grp = c("grp1", "grp1", "grp2", "grp2")))
corpus_group(corp, groups = grp)
corpus_group(corp, groups = c(1, 1, 2, 2))
corpus_group(corp, groups = factor(c(1, 1, 2, 2), levels = 1:3))

# with fill
corpus_group(corp, groups = factor(c(1, 1, 2, 2), levels = 1:3), fill = TRUE)
```

---

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"),
  use_docvars = TRUE,
  verbose = FALSE,
  ...
)
```

Arguments

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

Value

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

Examples

```
# simple example
corp1 <- 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)))
summary(corp1)
summary(corpus_reshape(corp1, to = "sentences"))

# example with inaugural corpus speeches
(corp2 <- corpus_subset(data_corpus_inaugural, Year>2004))
corp2para <- corpus_reshape(corp2, to = "paragraphs")
corp2para
summary(corp2para, 50, showmeta = TRUE)
## Note that Bush 2005 is recorded as a single paragraph because that text
## used a single \n to mark the end of a paragraph.
```

---

corpus_sample *Randomly sample documents from a corpus*

Description

Take a random sample of documents of the specified size from a corpus, with or without replacement, optionally by grouping variables or with probability weights.

Usage

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

Arguments

x	a corpus object whose documents will be sampled
size	a positive number, the number of documents to select; when used with by, the number to select from each group or a vector equal in length to the number of groups defining the samples to be chosen in each category of by. By defining a size larger than the number of documents, it is possible to oversample when replace = TRUE.
replace	if TRUE, sample with replacement
prob	a vector of probability weights for obtaining the elements of the vector being sampled. May not be applied when by is used.
by	optional grouping variable for sampling. This will be evaluated in the docvars data.frame, so that docvars may be referred to by name without quoting. This also changes previous behaviours for by. See news(Version >= "2.9", package = "quanteda") for details.

Value

a corpus object (re)sampled on the documents, containing the document variables for the documents sampled.

Examples

```
set.seed(123)
# sampling from a corpus
summary(corpus_sample(data_corpus_inaugural, size = 5))
summary(corpus_sample(data_corpus_inaugural, size = 10, replace = TRUE))

# sampling with by
corp <- data_corpus_inaugural
corp$century <- paste(floor(corp$Year / 100) + 1)
corp$century <- paste0(corp$century, ifelse(corp$century < 21, "th", "st"))
corpus_sample(corp, size = 2, by = century) |>
    summary()
# needs drop = TRUE to avoid empty interactions
corpus_sample(corp, size = 1, by = interaction(Party, century, drop = TRUE), replace = TRUE) |>
    summary()

# sampling sentences by document
corp <- corpus(c(one = "Sentence one.  Sentence two.  Third sentence.",
                 two = "First sentence, doc2.  Second sentence, doc2."),
               docvars = data.frame(var1 = c("a", "a"), var2 = c(1, 2)))
corpus_reshape(corp, to = "sentences") %>%
    corpus_sample(replace = TRUE, by = docid(.))

# oversampling
corpus_sample(corp, size = 5, replace = TRUE)
```

corpus_segment *Segment texts on a pattern match*

Description

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

Usage

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

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

Arguments

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

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

Details

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

Value

corpus_segment returns a corpus of segmented texts

char_segment returns a character vector of segmented texts

Boundaries and segmentation explained

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

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

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

Using patterns

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

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

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

```
Mr. Smith: Text
Mrs. Jones: More text
```

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

For custom boundary delimitation using punctuation characters that come come at the end of a clause or sentence (such as , and., these can be specified manually and pattern_position set to "after". To keep the punctuation characters in the text (as with sentence segmentation), set

extract_pattern = FALSE. (With most tag applications, users will want to remove the patterns from the text, as they are annotations rather than parts of the text itself.)

See Also

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

Examples

```
## segmenting a corpus

# segmenting a corpus using tags
corp1 <- corpus(c("##INTRO This is the introduction.
                  ##DOC1 This is the first document.  Second sentence in Doc 1.
                  ##DOC3 Third document starts here.  End of third document.",
                 "##INTRO Document ##NUMBER Two starts before ##NUMBER Three."))
corpseg1 <- corpus_segment(corp1, pattern = "##*")
cbind(corpseg1, docvars(corpseg1))

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

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

## segmenting a character vector

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

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

---

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, drop_docid = TRUE, ...)
```

Arguments

`x`	corpus object to be subsetted.
`subset`	logical expression indicating the documents to keep: missing values are taken as false.
`drop_docid`	if `TRUE`, `docid` for documents are removed as the result of subsetting.
`...`	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"))
```

---

corpus_trim *Remove sentences based on their token lengths or a pattern match*

Description

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

Usage

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

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

Arguments

x	corpus or character object whose sentences will be selected.
what	units of trimming, "sentences" or "paragraphs", or "documents"
min_ntoken, max_ntoken	minimum and maximum lengths in word tokens (excluding punctuation). Note that these are approximate numbers of tokens based on checking for word boundaries, rather than on-the-fly full tokenisation.
exclude_pattern	a **stringi** regular expression whose match (at the sentence level) will be used to exclude sentences

Value

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

Examples

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

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

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

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

Description

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

Usage

```
data_char_sampletext
```

Format

character vector with one element

Source

Dáil Éireann Debate, Financial Resolution No. 13: General (Resumed). https://www.oireachtas.ie/en/debates/debat
7 December 2011. vol. 749, no. 1.

Examples

```
tokens(data_char_sampletext, remove_punct = TRUE)
```

data_char_ukimmig2010 *Immigration-related sections of 2010 UK party manifestos*

Description

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

Usage

```
data_char_ukimmig2010
```

Format

A named character vector of plain ASCII texts

Examples

```
data_corpus_ukimmig2010 <-
    corpus(data_char_ukimmig2010,
           docvars = data.frame(party = names(data_char_ukimmig2010)))
summary(data_corpus_ukimmig2010, showmeta = TRUE)
```

`data_corpus_inaugural` *US presidential inaugural address texts*

Description

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

Usage

```
data_corpus_inaugural
```

Format

a corpus object with the following docvars:

- `Year` a four-digit integer year
- `President` character; President's last name
- `FirstName` character; President's first name (and possibly middle initial)
- `Party` factor; name of the President's political party

Details

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

Source

https://archive.org/details/Inaugural-Address-Corpus-1789-2009 and https://www.presidency.ucsb.edu/documents/presidential-documents-archive-guidebook/inaugural-addresses.

Examples

```
# some operations on the inaugural corpus
summary(data_corpus_inaugural)
head(docvars(data_corpus_inaugural), 10)
```

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, M., Benoit, K.R., & Garry, J. (2003). Estimating Policy Positions from Political Text using Words as Data. *American Political Science Review*, 97(2), 311–331.

data_dictionary_LSD2015

Lexicoder Sentiment Dictionary (2015)

Description

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

Usage

```
data_dictionary_LSD2015
```

Format

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

`negative` 2,858 word patterns indicating negative sentiment

`positive` 1,709 word patterns indicating positive sentiment

`neg_positive` 1,721 word patterns indicating a positive word preceded by a negation (used to convey negative sentiment)

`neg_negative` 2,860 word patterns indicating a negative word preceded by a negation (used to convey positive sentiment)

Details

The dictionary consists of 2,858 "negative" sentiment words and 1,709 "positive" sentiment words. A further set of 2,860 and 1,721 negations of negative and positive words, respectively, is also included. While many users will find the non-negation sentiment forms of the LSD adequate for sentiment analysis, Young and Soroka (2012) did find a small, but non-negligible increase in performance when accounting for negations. Users wishing to test this or include the negations are encouraged to subtract negated positive words from the count of positive words, and subtract the negated negative words from the negative count.

Young and Soroka (2012) also suggest the use of a pre-processing script to remove specific cases of some words (i.e., "good bye", or "nobody better", which should not be counted as positive). Pre-processing scripts are available at `https://www.snsoroka.com/data-lexicoder/`.

License and Conditions

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

Please cite the references below when using the dictionary.

References

The objectives, development and reliability of the dictionary are discussed in detail in Young and Soroka (2012). Please cite this article when using the Lexicoder Sentiment Dictionary and related resources. Young, L. & Soroka, S. (2012). *Lexicoder Sentiment Dictionary*. Available at `https://www.snsoroka.com/data-lexicoder/`.

Young, L. & Soroka, S. (2012). Affective News: The Automated Coding of Sentiment in Political Texts. doi:10.1080/10584609.2012.671234. *Political Communication*, 29(2), 205–231.

Examples

```
# simple example
txt <- "This aggressive policy will not win friends."

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

```
# notice that double-counting of negated and non-negated terms is avoided
# when using nested_scope = "dictionary"
tokens_lookup(tokens(txt), dictionary = data_dictionary_LSD2015,
              exclusive = FALSE, nested_scope = "dictionary")
## tokens from 1 document.
## text1 :
## [1] "This"   "NEGATIVE"   "policy"   "will"   "NEG_POSITIVE" "POSITIVE."

# compound neg_negative and neg_positive tokens before creating a dfm object
toks <- tokens_compound(tokens(txt), data_dictionary_LSD2015)

dfm_lookup(dfm(toks), data_dictionary_LSD2015)
```

`dfm` *Create a document-feature matrix*

Description

Construct a sparse document-feature matrix from a tokens or dfm object.

Usage

```
dfm(
  x,
  tolower = TRUE,
  remove_padding = FALSE,
  trim = TRUE,
  verbose = quanteda_options("verbose"),
  ...
)
```

Arguments

`x`	a tokens or dfm object.
`tolower`	convert all features to lowercase.
`remove_padding`	logical; if `TRUE`, remove the "pads" left as empty tokens after calling `tokens()` or `tokens_remove()` with `padding = TRUE`.
`trim`	logical; if `TRUE`, remove columns for features with all zeros. This is always `FALSE` when `x` records dictionary keys.
`verbose`	display messages if `TRUE`.
`...`	not used.

Details

The total number of non-zero values in `dfm` is constrained by the size of available space in RAM. If **quanteda** cannot allocate enough memory to construct a `dfm` from `tokens` object, users should apply `tokens_remove()` or `tokens_trim()` to reduce the size of the `tokens` objects before applying `dfm()`.

Value

a dfm object

Changes in version 3

In **quanteda** v4, many convenience functions formerly available in `dfm()` were removed.

See Also

`as.dfm()`, `dfm_select()`, dfm

Examples

```
## for a corpus
toks <- data_corpus_inaugural |>
  corpus_subset(Year > 1980) |>
  tokens()
dfm(toks)

# removal options
toks <- tokens(c("a b c", "A B C D")) |>
    tokens_remove("b", padding = TRUE)
toks
dfm(toks)
dfm(toks) |>
 dfm_remove(pattern = "") # remove "pads"

# preserving case
dfm(toks, tolower = FALSE)
```

---

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

Description

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

Usage

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

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.
verbose	if TRUE print a summary of the object returned by the function: the number of documents and, depending on the object, the number of characters, tokens and types, or features and occurrences. The number of tokens includes paddings, but the number of features excludes them.

Value

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

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

Note

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

Examples

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

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

# compress an fcm
fcmat1 <- fcm(tokens("A D a C E a d F e B A C E D"),
              context = "window", window = 3)
## this will produce an error:
# fcm_compress(fcmat1)

txt <- c("The fox JUMPED over the dog.",
         "The dog jumped over the fox.")
toks <- tokens(txt, remove_punct = TRUE)
fcmat2 <- fcm(toks, context = "document")
colnames(fcmat2) <- rownames(fcmat2) <- tolower(colnames(fcmat2))
colnames(fcmat2)[5] <- rownames(fcmat2)[5] <- "fox"
fcmat2
```

```
fcm_compress(fcmat2)
```

`dfm_group` *Combine documents in a dfm by a grouping variable*

Description

Combine documents in a dfm by a grouping variable, by summing the cell frequencies within group and creating new "documents" with the group labels.

Usage

```
dfm_group(
  x,
  groups = docid(x),
  fill = FALSE,
  force = FALSE,
  verbose = quanteda_options("verbose")
)
```

Arguments

`x`	a dfm
`groups`	grouping variable for sampling, equal in length to the number of documents. This will be evaluated in the docvars data.frame, so that docvars may be referred to by name without quoting. This also changes previous behaviours for `groups`. See `news(Version >= "3.0", package = "quanteda")` for details.
`fill`	logical; if `TRUE` and `groups` is a factor, then use all levels of the factor when forming the new documents of the grouped object. This will result in a new "document" with empty content for levels not observed, but for which an empty document may be needed. If `groups` is a factor of dates, for instance, then `fill = TRUE` ensures that the new object will consist of one new "document" by date, regardless of whether any documents previously existed with that date. Has no effect if the `groups` variable(s) are not factors.
`force`	logical; if `TRUE`, group by summing existing counts, even if the dfm has been weighted. This can result in invalid sums, such as adding log counts (when a dfm has been weighted by `"logcount"` for instance using `dfm_weight()`). Not needed when the term weight schemes "count" and "prop".
`verbose`	if `TRUE` print a summary of the object returned by the function: the number of documents and, depending on the object, the number of characters, tokens and types, or features and occurrences. The number of tokens includes paddings, but the number of features excludes them.

Value

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

Examples

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

# with fill = TRUE
dfm_group(dfmat, fill = TRUE,
          groups = factor(c("A", "A", "B", "C"), levels = LETTERS[1:4]))
```

---

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

Usage

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

Arguments

x	the dfm to which the dictionary will be applied
dictionary	a dictionary-class object

levels	levels of entries in a hierarchical dictionary that will be applied
exclusive	if TRUE, remove all features not in dictionary, otherwise, replace values in dictionary with keys while leaving other features unaffected
valuetype	the type of pattern matching: "glob" for "glob"-style wildcard expressions; "regex" for regular expressions; or "fixed" for exact matching. See valuetype for details.
case_insensitive	logical; if TRUE, ignore case when matching a pattern or dictionary values
capkeys	if TRUE, convert dictionary keys to uppercase to distinguish them from other features
nomatch	an optional character naming a new feature that will contain the counts of features of x not matched to a dictionary key. If NULL (default), do not tabulate unmatched features.
verbose	print status messages if TRUE

Note

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

See Also

dfm_replace

Examples

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

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

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

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

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

`dfm_match` *Match the dfm columns with given features*

Description

Match the columns of multiple dfm objects using a character vector.

Usage

```
dfm_match(x, features, verbose = quanteda_options("verbose"))
```

Arguments

`x`	the dfm object.
`features`	character vector for the feature names to be matched in the resulting dfm. Columns not included in `features` are removed.
`verbose`	if `TRUE` print a summary of the object returned by the function: the number of documents and, depending on the object, the number of characters, tokens and types, or features and occurrences. The number of tokens includes paddings, but the number of features excludes them.

Details

Matching the dfm columns is necessary when you split a corpus into two: you fit a model on the test set and evaluation it on the test set whose features must be identical. It is also used in `bootstrap_dfm()`.

Value

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

Note

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

See Also

`dfm_select()`

Examples

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

# matching one dfm to another
txt <- c("This is text one", "The text two", "This is text three")
(dfmt1 <- dfm(tokens(txt[1:2])))
(dfmt2 <- dfm(tokens(txt[2:3])))
(dfmt3 <- dfm(dfm_match(dfmt1, featnames(dfmt2))))
identical(featnames(dfmt2), featnames(dfmt3))
```

`dfm_replace` *Replace features in dfm*

Description

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

Usage

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

Arguments

`x`	dfm whose features will be replaced
`pattern`	a character vector. See pattern for more details.
`replacement`	if `pattern` is a character vector, then `replacement` must be character vector of equal length, for a 1:1 match.
`case_insensitive`	logical; if `TRUE`, ignore case when matching a `pattern` or dictionary values
`verbose`	if `TRUE` print a summary of the object returned by the function: the number of documents and, depending on the object, the number of characters, tokens and types, or features and occurrences. The number of tokens includes paddings, but the number of features excludes them.

Examples

```
dfmat1 <- dfm(tokens(data_corpus_inaugural))

# lemmatization
taxwords <- c("tax", "taxing", "taxed", "taxed", "taxation")
lemma <- rep("TAX", length(taxwords))
featnames(dfm_select(dfmat1, pattern = taxwords))
dfmat2 <- dfm_replace(dfmat1, pattern = taxwords, replacement = lemma)
featnames(dfm_select(dfmat2, pattern = taxwords))

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

---

dfm_sample *Randomly sample documents from a dfm*

Description

Take a random sample of documents of the specified size from a dfm, with or without replacement, optionally by grouping variables or with probability weights.

Usage

```
dfm_sample(
  x,
  size = NULL,
  replace = FALSE,
  prob = NULL,
  by = NULL,
  verbose = quanteda_options("verbose")
)
```

Arguments

x	the dfm object whose documents will be sampled
size	a positive number, the number of documents to select; when used with by, the number to select from each group or a vector equal in length to the number of groups defining the samples to be chosen in each category of by. By defining a size larger than the number of documents, it is possible to oversample when replace = TRUE.
replace	if TRUE, sample with replacement
prob	a vector of probability weights for obtaining the elements of the vector being sampled. May not be applied when by is used.

by	optional grouping variable for sampling. This will be evaluated in the docvars data.frame, so that docvars may be referred to by name without quoting. This also changes previous behaviours for by. See `news(Version >= "2.9", package = "quanteda")` for details.
verbose	if TRUE print a summary of the object returned by the function: the number of documents and, depending on the object, the number of characters, tokens and types, or features and occurrences. The number of tokens includes paddings, but the number of features excludes them.

Value

a dfm object (re)sampled on the documents, containing the document variables for the documents sampled.

See Also

sample

Examples

```
set.seed(10)
dfmat <- dfm(tokens(c("a b c c d", "a a c c d d d", "a b b c")))
dfmat
dfm_sample(dfmat)
dfm_sample(dfmat, replace = TRUE)

# by groups
dfmat <- dfm(tokens(data_corpus_inaugural[50:58]))
dfm_sample(dfmat, by = Party, size = 2)
```

---

dfm_select *Select features from a dfm or fcm*

Description

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

Usage

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

```
  max_nchar = NULL,
  padding = FALSE,
  verbose = quanteda_options("verbose")
)

dfm_remove(x, ...)

dfm_keep(x, ...)

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

fcm_remove(x, ...)

fcm_keep(x, ...)
```

Arguments

x	the dfm or fcm object whose features will be selected
pattern	a character vector, list of character vectors, dictionary, or collocations object. See pattern for details.
selection	whether to keep or remove the features
valuetype	the type of pattern matching: "glob" for "glob"-style wildcard expressions; "regex" for regular expressions; or "fixed" for exact matching. See valuetype for details.
case_insensitive	logical; if TRUE, ignore case when matching a pattern or dictionary values
min_nchar, max_nchar	optional numerics specifying the minimum and maximum length in characters for tokens to be removed or kept; defaults are NULL for no limits. These are applied after (and hence, in addition to) any selection based on pattern matches.
padding	if TRUE, record the number of removed tokens in the first column.
verbose	if TRUE, print message about how many pattern were removed
...	used only for passing arguments from dfm_remove or dfm_keep to dfm_select. Cannot include selection.

Details

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

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

Value

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

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

Note

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

See Also

dfm_match()

Examples

```
dfmat <- tokens(c("My Christmas was ruined by your opposition tax plan.",
                  "Does the United_States or Sweden have more progressive taxation?")) |>
    dfm(tolower = FALSE)
dict <- dictionary(list(countries = c("United_States", "Sweden", "France"),
                        wordsEndingInY = c("by", "my"),
                        notintext = "blahblah"))
dfm_select(dfmat, pattern = dict)
dfm_select(dfmat, pattern = dict, case_insensitive = FALSE)
dfm_select(dfmat, pattern = c("s$", ".y"), selection = "keep", valuetype = "regex")
dfm_select(dfmat, pattern = c("s$", ".y"), selection = "remove", valuetype = "regex")
dfm_select(dfmat, pattern = stopwords("english"), selection = "keep", valuetype = "fixed")
dfm_select(dfmat, pattern = stopwords("english"), selection = "remove", valuetype = "fixed")

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

dfmat <- dfm(tokens(c("This is a document with lots of stopwords.",
                      "No if, and, or but about it: lots of stopwords.")))
dfmat
dfm_remove(dfmat, stopwords("english"))
toks <- tokens(c("this contains lots of stopwords",
                 "no if, and, or but about it: lots"),
               remove_punct = TRUE)
fcmat <- fcm(toks)
fcmat
fcm_remove(fcmat, 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	logical; if TRUE, the sort will be in descending order, otherwise sort in increasing order
margin	which margin to sort on features to sort by frequency of features, documents to sort by total feature counts in documents, and both to sort by both

Value

A sorted dfm matrix object

Author(s)

Ken Benoit

Examples

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

dfm_subset *Extract a subset of a dfm*

Description

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

Usage

```
dfm_subset(
  x,
  subset,
  min_ntoken = NULL,
  max_ntoken = NULL,
  drop_docid = TRUE,
  verbose = quanteda_options("verbose"),
  ...
)
```

Arguments

x	dfm object to be subsetted.
subset	logical expression indicating the documents to keep: missing values are taken as false.
min_ntoken, max_ntoken	minimum and maximum lengths of the documents to extract.
drop_docid	if TRUE, docid for documents are removed as the result of subsetting.
verbose	if TRUE print a summary of the object returned by the function: the number of documents and, depending on the object, the number of characters, tokens and types, or features and occurrences. The number of tokens includes paddings, but the number of features excludes them.
...	not used

Details

To select or subset *features*, see dfm_select() instead.

Value

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

Examples

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

dfm_tfidf *Weight a dfm by* tf-idf

Description

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

Usage

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

Arguments

x	object for which idf or tf-idf will be computed (a document-feature matrix)
scheme_tf	scheme for dfm_weight(); defaults to "count"
scheme_df	scheme for docfreq(); defaults to "inverse".
base	the base for the logarithms in the dfm_weight() and docfreq() calls; default is 10
force	logical; if TRUE, apply weighting scheme even if the dfm has been weighted before. This can result in invalid weights, such as as weighting by "prop" after applying "logcount", or after having grouped a dfm using dfm_group().
...	additional arguments passed to docfreq.

Details

dfm_tfidf computes term frequency-inverse document frequency weighting. The default is to use counts instead of normalized term frequency (the relative term frequency within document), but this can be overridden using scheme_tf = "prop".

References

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge: Cambridge University Press. https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf

Examples

```
dfmat1 <- as.dfm(data_dfm_lbgexample)
head(dfmat1[, 5:10])
head(dfm_tfidf(dfmat1)[, 5:10])
docfreq(dfmat1)[5:15]
head(dfm_weight(dfmat1)[, 5:10])

# replication of worked example from
# https://en.wikipedia.org/wiki/Tf-idf#Example_of_tf.E2.80.93idf
dfmat2 <-
    matrix(c(1,1,2,1,0,0, 1,1,0,0,2,3),
           byrow = TRUE, nrow = 2,
           dimnames = list(docs = c("document1", "document2"),
                           features = c("this", "is", "a", "sample",
                                        "another", "example"))) |>
    as.dfm()
dfmat2
docfreq(dfmat2)
dfm_tfidf(dfmat2, scheme_tf = "prop") |> round(digits = 2)

## Not run:
# comparison with tm
if (requireNamespace("tm")) {
    convert(dfmat2, to = "tm") |> tm::weightTfIdf() |> as.matrix()
    # same as:
    dfm_tfidf(dfmat2, base = 2, scheme_tf = "prop")
}

## End(Not run)
```

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

Description

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

Usage

```
dfm_tolower(x, keep_acronyms = FALSE, verbose = quanteda_options("verbose"))

dfm_toupper(x, verbose = quanteda_options("verbose"))

fcm_tolower(x, keep_acronyms = FALSE, verbose = quanteda_options("verbose"))

fcm_toupper(x, verbose = quanteda_options("verbose"))
```

Arguments

x	the input object whose character/tokens/feature elements will be case-converted
keep_acronyms	logical; if TRUE, do not lowercase any all-uppercase words (applies only to *_tolower() functions)
verbose	if TRUE print a summary of the object returned by the function: the number of documents and, depending on the object, the number of characters, tokens and types, or features and occurrences. The number of tokens includes paddings, but the number of features excludes them.

Details

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

Examples

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

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

---

dfm_trim *Trim a dfm using frequency threshold-based feature selection*

Description

Returns a document by feature matrix reduced in size based on document and term frequency, usually in terms of a minimum frequency, but may also be in terms of maximum frequencies. Setting a combination of minimum and maximum frequencies will select features based on a range.

Feature selection is implemented by considering features across all documents, by summing them for term frequency, or counting the documents in which they occur for document frequency. Rank and quantile versions of these are also implemented, for taking the first n features in terms of descending order of overall global counts or document frequencies, or as a quantile of all frequencies.

Usage

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

Arguments

x	a dfm object
min_termfreq, max_termfreq	minimum/maximum values of feature frequencies across all documents, below/above which features will be removed
termfreq_type	how min_termfreq and max_termfreq are interpreted. "count" sums the frequencies; "prop" divides the term frequencies by the total sum; "rank" is matched against the inverted ranking of features in terms of overall frequency, so that 1, 2, ... are the highest and second highest frequency features, and so on; "quantile" sets the cutoffs according to the quantiles (see quantile()) of term frequencies.
min_docfreq, max_docfreq	minimum/maximum values of a feature's document frequency, below/above which features will be removed
docfreq_type	specify how min_docfreq and max_docfreq are interpreted. "count" is the same as [docfreq](x, scheme = "count"); "prop" divides the document frequencies by the total sum; "rank" is matched against the inverted ranking of document frequency, so that 1, 2, ... are the features with the highest and second highest document frequencies, and so on; "quantile" sets the cutoffs according to the quantiles (see quantile()) of document frequencies.
max_n	the maximum number of features. Features are selected based on their frequencies, but the earlier elements in featnames or types are chosen when the frequencies are equal.
sparsity	equivalent to 1 - min_docfreq, included for comparison with **tm**
verbose	if TRUE print a summary of the object returned by the function: the number of documents and, depending on the object, the number of characters, tokens and types, or features and occurrences. The number of tokens includes paddings, but the number of features excludes them.

Value

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

Note

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

See Also

`dfm_select()`, `dfm_sample()`

Examples

```
dfmat <- dfm(tokens(data_corpus_inaugural))

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

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

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

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

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

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

## End(Not run)
```

---

dfm_weight *Weight the feature frequencies in a dfm*

Description

Weight the feature frequencies in a dfm

Usage

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

dfm_smooth(x, smoothing = 1)
```

Arguments

x	document-feature matrix created by dfm
scheme	a label of the weight type:
	`count` tf_{ij}, an integer feature count (default when a dfm is created)
	`prop` the proportion of the feature counts of total feature counts (aka relative frequency), calculated as $tf_{ij} / \sum_j tf_{ij}$
	`propmax` the proportion of the feature counts of the highest feature count in a document, $tf_{ij} / \max_j tf_{ij}$
	`logcount` take the 1 + the logarithm of each count, for the given base, or 0 if the count was zero: $1 + \log_{base}(tf_{ij})$ if $tf_{ij} > 0$, or 0 otherwise.
	`boolean` recode all non-zero counts as 1
	`augmented` equivalent to $k + (1 - k) *$ `dfm_weight(x, "propmax")`
	`logave` (1 + the log of the counts) / (1 + log of the average count within document), or $$\frac{1 + \log_{base} tf_{ij}}{1 + \log_{base}(\sum_j tf_{ij} / N_i)}$$
	`logsmooth` log of the counts + `smooth`, or $tf_{ij} + s$
weights	if `scheme` is unused, then `weights` can be a named numeric vector of weights to be applied to the dfm, where the names of the vector correspond to feature labels of the dfm, and the weights will be applied as multipliers to the existing feature counts for the corresponding named features. Any features not named will be assigned a weight of 1.0 (meaning they will be unchanged).
base	base for the logarithm when `scheme` is `"logcount"` or `logave`
k	the k for the augmentation when `scheme = "augmented"`
smoothing	constant added to the dfm cells for smoothing, default is 1 for `dfm_smooth()` and 0.5 for `dfm_weight()`.
force	logical; if `TRUE`, apply weighting scheme even if the dfm has been weighted before. This can result in invalid weights, such as as weighting by `"prop"` after applying `"logcount"`, or after having grouped a dfm using `dfm_group()`.

Value

`dfm_weight` returns the dfm with weighted values. Note the because the default weighting scheme is `"count"`, simply calling this function on an unweighted dfm will return the same object. Many users will want the normalized dfm consisting of the proportions of the feature counts within each document, which requires setting `scheme = "prop"`.

`dfm_smooth` returns a dfm whose values have been smoothed by adding the `smoothing` amount. Note that this effectively converts a matrix from sparse to dense format, so may exceed memory requirements depending on the size of your input matrix.

References

Manning, C.D., Raghavan, P., & Schütze, H. (2008). *An Introduction to Information Retrieval*. Cambridge: Cambridge University Press. https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf

See Also

`docfreq()`

Examples

```
dfmat1 <- dfm(tokens(data_corpus_inaugural))

dfmat2 <- dfm_weight(dfmat1, scheme = "prop")
topfeatures(dfmat2)
dfmat3 <- dfm_weight(dfmat1)
topfeatures(dfmat3)
dfmat4 <- dfm_weight(dfmat1, scheme = "logcount")
topfeatures(dfmat4)
dfmat5 <- dfm_weight(dfmat1, scheme = "logave")
topfeatures(dfmat5)

# combine these methods for more complex dfm_weightings, e.g. as in Section 6.4
# of Introduction to Information Retrieval
head(dfm_tfidf(dfmat1, scheme_tf = "logcount"))

# smooth the dfm
dfmat <- dfm(tokens(data_corpus_inaugural))
dfm_smooth(dfmat, 0.5)
```

---

dictionary *Create a dictionary object*

Description

Create a **quanteda** dictionary object to perform pattern matching on tokens, dfm and fcm.

Usage

```
dictionary(
  x,
  file = NULL,
  format = NULL,
  separator = " ",
  tolower = TRUE,
  tokenize = FALSE,
  levels = 1:100,
  encoding = "utf-8"
)
```

Arguments

x	a named list of valuetype patterns or an existing dictionary object. See examples. This argument should be omitted if `file` is specified.
file	file identifier for a foreign dictionary.
format	character identifier for the format of the foreign dictionary. If not supplied, the format is guessed from the dictionary file's extension. Available options are: `"wordstat"` format used by Provalis Research's WordStat software `"LIWC"` format used by the Linguistic Inquiry and Word Count software `"yoshikoder"` format used by Yoshikoder software `"lexicoder"` format used by Lexicoder `"YAML"` the standard YAML format
separator	the character in between multi-word dictionary values. This defaults to `" "`.
tolower	if `TRUE`, convert all dictionary values to lowercase.
tokenize	if `TRUE` segment dictionary values by separators to using the built-in tokenizer. Useful for Japanese and Chinese dictionaries.
levels	integers specifying the levels of entries in `x` or `file` to be included in the object.
encoding	additional optional encoding value for reading in imported dictionaries. This uses the iconv labels for encoding. See the "Encoding" section of the help for file.

Details

A dictionary object can include multi-word expressions segmented by `separator`. When it is applied to tokens object, they match both sequences of separate tokens and compounded tokens.

Dictionary objects can be subsetted using `[` and `[[`, operating the same as the equivalent list operators. If `dictionary()` is applied to existing objects, it is possible to select `levels`.

Dictionary objects can be coerced from and to lists using `as.dictionary()` and `as.list()`, and checked using `is.dictionary()`.

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).

Value

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

References

WordStat dictionaries page, from Provalis Research https://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 (https://www.liwc.app/).

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

Lexicoder format, https://www.snsoroka.com/data-lexicoder/

See Also

as.dictionary(), as.list(), is.dictionary()

Examples

```
corp <- corpus_subset(data_corpus_inaugural, Year > 2000)
toks <- tokens(corp)

dict <- dictionary(list(
   tax = c("tax", "taxes", "taxing"),         # fixed patterns
   economy = list("econom*",                  # glob patterns
                  job = c("work*", "job*")),  # nested keys
   health = c("health care", "public health") # multi-word expressions
))

# compound tokens
tokens_compound(toks, pattern = dict) |>
    dfm() |>
    dfm_select(dict)

tokens_lookup(toks, dictionary = dict, levels = 1) |>
    dfm()

# subset a dictionary
dict[1:2]
dict[c("economy")]

# update a dictionary
dictionary(dict, levels = 2)

## Not run:
dfmat <- dfm(tokens(data_corpus_inaugural))

# import the Laver-Garry dictionary from Provalis Research
download.file("https://provalisresearch.com/Download/LaverGarry.zip",
              tf <- tempfile(), mode = "wb")
```

```
unzip(tf, exdir = (td <- tempdir()))
dict_lg <- dictionary(file = paste(td, "LaverGarry.cat", sep = "/"))
dfm_lookup(dfmat, dict_lg)

# import a LIWC formatted dictionary from http://www.moralfoundations.org
download.file("http://bit.ly/37cV95h", tf <- tempfile())
dict_liwc <- dictionary(file = tf, format = "LIWC")
dfm_lookup(dfmat, dict_liwc)

## End(Not run)
```

`docfreq` *Compute the (weighted) document frequencies of features*

Description

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

Usage

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

Arguments

`x`	a dfm, tokens, or tokens_xptr object.
`scheme`	type of document frequency weighting, computed as follows, where N is defined as the number of documents in `x` and s is the smoothing constant:
	`count` df_j, the number of documents for which $n_{ij} > threshold$
	`inverse` $\log_{base}\left(s + \frac{N}{k + df_j}\right)$
	`inversemax` $\log_{base}\left(s + \frac{\max(df_j)}{k + df_j}\right)$
	`inverseprob` $\log_{base}\left(\frac{N - df_j}{k + df_j}\right)$
	`unary` 1 for each feature

base	the base with respect to which logarithms in the inverse document frequency weightings are computed; default is 10 (see Manning, Raghavan, and Schütze 2008, p123).
smoothing	added to the quotient before taking the logarithm
k	added to the denominator in the "inverse" weighting types, to prevent a zero document count for a term
threshold	numeric value of the threshold *above which* a feature will considered in the computation of document frequency. The default is 0, meaning that a feature's document frequency will be the number of documents in which it occurs greater than zero times. Only supported when x is a dfm; for tokens and tokens_xptr objects it must be 0 (the default).

Value

a numeric vector of document frequencies for each feature

References

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge: Cambridge University Press. https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf

Examples

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

# replication of worked example from
# https://en.wikipedia.org/wiki/Tf-idf#Example_of_tf.E2.80.93idf
dfmat2 <-
    matrix(c(1,1,2,1,0,0, 1,1,0,0,2,3),
           byrow = TRUE, nrow = 2,
           dimnames = list(docs = c("document1", "document2"),
                           features = c("this", "is", "a", "sample",
                                        "another", "example"))) |>
    as.dfm()
dfmat2
docfreq(dfmat2)
docfreq(dfmat2, scheme = "inverse")
docfreq(dfmat2, scheme = "inverse", k = 1, smoothing = 1)
docfreq(dfmat2, scheme = "unary")
docfreq(dfmat2, scheme = "inversemax")
docfreq(dfmat2, scheme = "inverseprob")
```

docnames *Get or set document names*

Description

Get or set document names

Usage

```
docnames(x)

docnames(x, unique_docnames = TRUE) <- value

docid(x)

segid(x)
```

Arguments

x	corpus, tokens, tokens_xptr, or dfm object.
unique_docnames	logical; if TRUE, enforce strict uniqueness in docnames; otherwise, rename duplicated docnames using an added serial number, and treat them as segments of the same document.
value	a character vector of the same length as x

Value

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

`docid()` returns an internal factor variable that records the original `doc_id`. If an object has been reshaped (e.g. `corpus_reshape()` or segmented (e.g. `corpus_segment()`), `docid(x)` returns the original `doc_id` but `segid(x)` does the serial number of those segments within the original `doc_id`.

Note

`docid()` and `segid()` are designed primarily for developers, not for end users. In most cases, you will want `docnames()` instead. It is, however, the default for groups, so that documents that have been previously reshaped (e.g. `corpus_reshape()` or segmented (e.g. `corpus_segment()`) will be regrouped into their original `doc_id` when `groups = docid(x)`.

See Also

`featnames()`

Examples

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

# get and set doument names to a tokens
toks <- tokens(corp)
docnames(toks) <- char_tolower(docnames(toks))

# get and set doument names to a dfm
dfmat <- dfm(tokens(corp))
docnames(dfmat) <- char_tolower(docnames(dfmat))

# reassign the document names of the inaugural speech corpus
corp <- data_corpus_inaugural
docnames(corp) <- paste0("Speech", seq_len(ndoc(corp)))


corp <- corpus(c(textone = "This is a sentence.  Another sentence.  Yet another.",
                 textwo = "Sentence 1. Sentence 2."))
corp_sent <- corp |>
    corpus_reshape(to = "sentences")
docnames(corp_sent)

# docid
docid(corp_sent)
docid(tokens(corp_sent))
docid(dfm(tokens(corp_sent)))

# segid
segid(corp_sent)
segid(tokens(corp_sent))
segid(dfm(tokens(corp_sent)))
```

`docvars` *Get or set document-level variables*

Description

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

Usage

```
docvars(x, field = NULL)

docvars(x, field = NULL) <- value

## S3 method for class 'corpus'
x$name
```

```
## S3 replacement method for class 'corpus'
x$name <- value

## S3 method for class 'tokens'
x$name

## S3 replacement method for class 'tokens'
x$name <- value

## S3 method for class 'dfm'
x$name

## S3 replacement method for class 'dfm'
x$name <- 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	a vector of document variable values to be assigned to name
name	a literal character string specifying a single docvars name

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

Accessing or assigning docvars using the $ operator

As of **quanteda** v2, it is possible to access and assign a docvar using the $ operator. See Examples.

Note

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

Examples

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

# assigning document variables to a corpus
```

```
corp <- data_corpus_inaugural
docvars(corp, "President") <- paste("prez", 1:ndoc(corp), sep = "")
head(docvars(corp))
corp$fullname <- paste(data_corpus_inaugural$FirstName,
                       data_corpus_inaugural$President)
tail(corp$fullname)


# accessing or assigning docvars for a corpus using "$"
data_corpus_inaugural$Year
data_corpus_inaugural$century <- floor(data_corpus_inaugural$Year / 100)
data_corpus_inaugural$century

# accessing or assigning docvars for tokens using "$"
toks <- tokens(corpus_subset(data_corpus_inaugural, Year <= 1805))
toks$Year
toks$Year <- 1991:1995
toks$Year
toks$nonexistent <- TRUE
docvars(toks)

# accessing or assigning docvars for a dfm using "$"
dfmat <- dfm(toks)
dfmat$Year
dfmat$Year <- 1991:1995
dfmat$Year
dfmat$nonexistent <- TRUE
docvars(dfmat)
```

`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 = NULL,
  ordered = FALSE,
  tri = TRUE,
  ...
)
```

Arguments

x	a tokens, or dfm object from which to generate the feature co-occurrence matrix
context	the context in which to consider term co-occurrence: "document" for co-occurrence counts within document; "window" for co-occurrence within a defined window of words, which requires a positive integer value for window. Note: if x is a dfm object, then context can only be "document".
count	how to count co-occurrences: "frequency" count the number of co-occurrences within the context "boolean" count only the co-occurrence or not within the context, irrespective of how many times it occurs. "weighted" count a weighted function of counts, typically as a function of distance from the target feature. Only makes sense for context = "window".
window	positive integer value for the size of a window on either side of the target feature, default is 5, meaning 5 words before and after the target feature
weights	a vector of weights applied to each distance from 1:window, strictly decreasing by default; can be a custom-defined vector of the same length as window
ordered	if TRUE, count only the forward co-occurrences for each target token for bigram models, so that the i, j cell of the fcm is the number of times that token j occurs before the target token i within the window. Only makes sense for context = "window", and when ordered = TRUE, the argument tri has no effect.
tri	if TRUE return only upper triangle (including diagonal). Ignored if ordered = TRUE.
...	not used here

Details

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

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

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

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

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

Author(s)

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

References

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, 325-328. https://aclanthology.org/N10-1046/

Jurafsky, D. & Martin, J.H. (2018). From *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Draft of September 23, 2018 (Chapter 6, Vector Semantics). Available at https://web.stanford.edu/~jurafsky/slp3/.

Church, K. W. & P. Hanks (1990). Word association norms, mutual information, and lexicography. *Computational Linguistics*, 16(1), 22-29. https://aclanthology.org/J90-1003/

Examples

```
# see http://bit.ly/29b2zOA
toks1 <- tokens(c("A D A C E A D F E B A C E D"))
fcm(toks1, context = "window", window = 2)
fcm(toks1, context = "window", count = "weighted", window = 3)
fcm(toks1, context = "window", count = "weighted", window = 3,
    weights = c(3, 2, 1), ordered = TRUE, tri = FALSE)

# with multiple documents
toks2 <- tokens(c("a a a b b c", "a a c e", "a c e f g"))
fcm(toks2, context = "document", count = "frequency")
fcm(toks2, context = "document", count = "boolean")
fcm(toks2, context = "window", window = 2)

txt3 <- c("The quick brown fox jumped over the lazy dog.",
         "The dog jumped and ate the fox.")
toks3 <- tokens(char_tolower(txt3), remove_punct = TRUE)
fcm(toks3, context = "document")
fcm(toks3, context = "window", window = 3)
```

---

fcm_sort *Sort an fcm in alphabetical order of the features*

Description

Sorts an fcm in alphabetical order of the features.

Usage

```
fcm_sort(x)
```

Arguments

x	fcm object

Value

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

Author(s)

Kenneth Benoit

Examples

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

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

`featfreq` *Compute the frequencies of features*

Description

Returns a frequency for each feature, computed across all documents. This is equivalent to `colSums(x)` when `x` is a dfm.

Usage

```
featfreq(x)
```

Arguments

x	a dfm, tokens, or tokens_xptr object.

Value

a (named) numeric vector of feature frequencies.

See Also

`dfm_tfidf()`, `dfm_weight()`

Examples

```
dfmat <- dfm(tokens(data_char_sampletext))
featfreq(dfmat)
```

featnames *Get the feature labels from a dfm*

Description

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

Usage

```
featnames(x)
```

Arguments

x	the dfm whose features will be extracted

Value

character vector of the feature labels

Examples

```
dfmat <- dfm(tokens(data_corpus_inaugural))

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

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

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

index *Locate a pattern in a tokens object*

Description

Locates a pattern within a tokens object, returning the index positions of the beginning and ending tokens in the pattern.

Usage

```
index(
  x,
  pattern,
  valuetype = c("glob", "regex", "fixed"),
  case_insensitive = TRUE
)

is.index(x)
```

Arguments

x	an input tokens object
pattern	a character vector, list of character vectors, dictionary, or collocations object. See pattern for details.
valuetype	the type of pattern matching: "glob" for "glob"-style wildcard expressions; "regex" for regular expressions; or "fixed" for exact matching. See valuetype for details.
case_insensitive	logical; if TRUE, ignore case when matching a pattern or dictionary values

Value

a data.frame consisting of one row per pattern match, with columns for the document name, index positions from and to, and the pattern matched.

is.index returns TRUE if the object was created by index(); FALSE otherwise.

Examples

```
toks <- tokens(data_corpus_inaugural[1:8])
index(toks, pattern = "secure*")
index(toks, pattern = c("secure*", phrase("united states"))) |> head()
```

---

is.collocations *Check if an object is collocations*

Description

Function to check if an object is a collocations object, created by quanteda.textstats::textstat_collocations().

Usage

```
is.collocations(x)
```

Arguments

x	object to be checked

Value

TRUE if the object is of class collocations, FALSE otherwise

---

kwic *Locate keywords-in-context*

Description

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

Usage

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

is.kwic(x)

## S3 method for class 'kwic'
as.data.frame(x, ...)
```

Arguments

x	a character, corpus, or tokens object
pattern	a character vector, list of character vectors, dictionary, or collocations object. See pattern for details.
window	the number of context words to be displayed around the keyword
valuetype	the type of pattern matching: "glob" for "glob"-style wildcard expressions; "regex" for regular expressions; or "fixed" for exact matching. See valuetype for details.
separator	a character to separate words in the output
case_insensitive	logical; if TRUE, ignore case when matching a pattern or dictionary values
index	an index object to specify keywords
...	unused

Value

A kwic classed data.frame, with the document name (docname) and the token index positions (from and to, which will be the same for single-word patterns, or a sequence equal in length to the number of elements for multi-word phrases).

Note

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

See Also

print-methods

Examples

```
# single token matching
toks <- tokens(data_corpus_inaugural[1:8])
kwic(toks, pattern = "secure*", valuetype = "glob", window = 3)
kwic(toks, pattern = "secur", valuetype = "regex", window = 3)
kwic(toks, pattern = "security", valuetype = "fixed", window = 3)

# phrase matching
kwic(toks, pattern = phrase("secur* against"), window = 2)
kwic(toks, pattern = phrase("war against"), valuetype = "regex", window = 2)

# use index
idx <- index(toks, phrase("secur* against"))
kwic(toks, index = idx, window = 2)

kw <- kwic(tokens(data_corpus_inaugural[1:20]), "provident*")
is.kwic(kw)
is.kwic("Not a kwic")
is.kwic(kw[, c("pre", "post")])

toks <- tokens(data_corpus_inaugural[1:8])
kw <- kwic(toks, pattern = "secure*", valuetype = "glob", window = 3)
as.data.frame(kw)
```

---

meta *Get or set object metadata*

Description

Get or set the object metadata in a corpus, tokens, dfm, or dictionary object. With the exception of dictionaries, this will be corpus-level metadata.

Usage

```
meta(x, field = NULL, type = c("user", "object", "system", "all"))

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

Arguments

x	an object for which the metadata will be read or set
field	metadata field name(s); if NULL (default), return all metadata names
type	"user" for user-provided corpus-level metadata; "system" for metadata set automatically when the corpus is created; or "all" for all metadata.
value	new value of the metadata field

Value

For meta, a named list of the metadata fields in the corpus.

For meta <-, the corpus with the updated user-level metadata. Only user-level metadata may be assigned.

Examples

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

---

ndoc *Count the number of documents or features*

Description

Get the number of documents or features in an object.

Usage

```
ndoc(x)

nfeat(x)
```

Arguments

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

Value

ndoc() returns an integer count of the number of documents in an object whose texts are organized as "documents" (a corpus, dfm, or tokens/tokens_xptr object.

nfeat() returns an integer count of the number of features. It is an alias for ntype() for a dfm. This function is only defined for dfm objects because only these have "features".

See Also

ntoken(), ntype()

Examples

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

# number of features
toks1 <- tokens(corpus_subset(data_corpus_inaugural, Year > 1980), remove_punct = FALSE)
toks2 <- tokens(corpus_subset(data_corpus_inaugural, Year > 1980), remove_punct = TRUE)
nfeat(dfm(toks1))
nfeat(dfm(toks2))
```

---

nsentence *Count the number of sentences*

Description

[Deprecated]

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

Value

count(s) of the total sentences per text

Note

`nsentence()` is now deprecated for all usages except tokens objects that have already been tokenised with `tokens(x, what = "sentence")`. Using it on character or corpus objects will now generate a warning.

`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.")
tokens(txt, what = "sentence") |>
    nsentence()
```

`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** tokens or dfm object
`...`	additional arguments passed to `tokens()`

Value

`ntoken()` returns a named integer vector of the counts of the total tokens

`ntypes()` returns a named integer vector of the counts of the types (unique tokens) per document. For dfm objects, `ntype()` will only return the count of features that occur more than zero times in the dfm.

Examples

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

# with some real texts
toks <- tokens(corpus_subset(data_corpus_inaugural, Year < 1806))
ntoken(tokens(toks, remove_punct = TRUE))
ntype(tokens(toks, remove_punct = TRUE))
ntoken(dfm(toks))
ntype(dfm(toks))
```

---

phrase *Declare a pattern to be a sequence of separate patterns*

Description

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

Usage

```
phrase(x, separator = " ")

as.phrase(x)

is.phrase(x)
```

Arguments

`x`	character, dictionary, list, collocations, or tokens object; the compound patterns to be treated as a sequence separated by `separator`. For list, collocations, or tokens objects, use `as.phrase()`.
`separator`	character; the character in between the patterns. This defaults to " ". For `phrase()` only.

Value

`phrase()` and `as.phrase()` return a specially classed list whose elements have been split into separate `character` (pattern) elements.

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

See Also

as.phrase()

Examples

```
# make phrases from characters
phrase(c("natural language processing"))
phrase(c("natural_language_processing", "text_analysis"), separator = "_")

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

# from a list
as.phrase(list(c("natural", "language", "processing")))

# from tokens
as.phrase(tokens("natural language processing"))
```

---

print-methods *Print methods for quanteda core objects*

Description

Print method for **quanteda** objects. In each max_n* option, 0 shows none, and -1 shows all.

Usage

```
## S3 method for class 'corpus'
print(
  x,
  max_ndoc = quanteda_options("print_corpus_max_ndoc"),
  max_nchar = quanteda_options("print_corpus_max_nchar"),
  show_summary = quanteda_options("print_corpus_summary"),
  ...
)

## S4 method for signature 'dfm'
print(
  x,
  max_ndoc = quanteda_options("print_dfm_max_ndoc"),
  max_nfeat = quanteda_options("print_dfm_max_nfeat"),
  show_summary = quanteda_options("print_dfm_summary"),
  ...
)

## S4 method for signature 'dictionary2'
print(
  x,
```

```
  max_nkey = quanteda_options("print_dictionary_max_nkey"),
  max_nval = quanteda_options("print_dictionary_max_nval"),
  show_summary = quanteda_options("print_dictionary_summary"),
  ...
)

## S4 method for signature 'fcm'
print(
  x,
  max_nfeat = quanteda_options("print_dfm_max_nfeat"),
  show_summary = TRUE,
  ...
)

## S3 method for class 'kwic'
print(
  x,
  max_nrow = quanteda_options("print_kwic_max_nrow"),
  show_summary = quanteda_options("print_kwic_summary"),
  ...
)

## S3 method for class 'tokens'
print(
  x,
  max_ndoc = quanteda_options("print_tokens_max_ndoc"),
  max_ntoken = quanteda_options("print_tokens_max_ntoken"),
  show_summary = quanteda_options("print_tokens_summary"),
  ...
)
```

Arguments

x	the object to be printed
max_ndoc	max number of documents to print; default is from the print_*_max_ndoc setting of quanteda_options()
max_nchar	max number of tokens to print; default is from the print_corpus_max_nchar setting of quanteda_options()
show_summary	print a brief summary indicating the number of documents and other characteristics of the object, such as docvars or sparsity.
...	passed to base::print() for tokens; unused for all other objects.
max_nfeat	max number of features to print; default is from the print_dfm_max_nfeat setting of quanteda_options()
max_nkey	max number of keys to print; default is from the print_dictionary_max_max_nkey setting of quanteda_options()
max_nval	max number of values to print; default is from the print_dictionary_max_nval setting of quanteda_options()

max_nrow	max number of documents to print; default is from the print_kwic_max_nrow setting of quanteda_options()
max_ntoken	max number of tokens to print; default is from the print_tokens_max_ntoken setting of quanteda_options().

See Also

quanteda_options()

Examples

```
corp <- corpus(data_char_ukimmig2010)
print(corp, max_ndoc = 3, max_nchar = 40)

toks <- tokens(corp)
print(toks, max_ndoc = 3, max_ntoken = 6)

dfmat <- dfm(toks)
print(dfmat, max_ndoc = 3, max_nfeat = 10)
```

---

quanteda_options *Get or set package options for quanteda*

Description

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

Usage

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

Arguments

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

Details

Currently available options are:

verbose logical; if TRUE then use this as the default for all functions with a verbose argument

threads integer; specifies the number of threads to use in parallelized functions; defaults to the maximum number of threads

print_dfm_max_ndoc, print_corpus_max_ndoc, print_tokens_max_ndoc integer; specify the number of documents to display when using the defaults for printing a dfm, corpus, or tokens object

print_dfm_max_nfeat, print_corpus_max_nchar, print_tokens_max_ntoken integer; specifies the number of features to display when printing a dfm, the number of characters to display when printing corpus documents, or the number of tokens to display when printing tokens objects

print_dfm_summary integer; specifies the number of documents to display when using the defaults for printing a dfm

print_dictionary_max_nkey, print_dictionary_max_nval the number of keys or values (respectively) to display when printing a dictionary

print_kwic_max_nrow the number of rows to display when printing a kwic object

base_docname character; stem name for documents that are unnamed when a corpus, tokens, or dfm are created or when a dfm is converted from another object

base_featname character; stem name for features that are unnamed when they are added, for whatever reason, to a dfm through an operation that adds features

base_compname character; stem name for components that are created by matrix factorization

language_stemmer character; language option for char_wordstem(), tokens_wordstem(), and dfm_wordstem()

pattern_hashtag, pattern_username character; regex patterns for (social media) hashtags and usernames respectively, used to avoid segmenting these in the default internal "word" tokenizer

tokens_block_size integer; specifies the number of documents to be tokenized at a time in blocked tokenization. When the number is large, tokenization becomes faster but also memory-intensive.

tokens_locale character; specify locale in stringi boundary detection in tokenization and corpus reshaping. See stringi::stri_opts_brkiter().

tokens_tokenizer_word character; the current word tokenizer version used as a default for what = "word" in tokens(), one of "word1", "word2", "word3" (same as "word2"), or "word4".

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

```
(opt <- quanteda_options())

quanteda_options(verbose = TRUE)
quanteda_options("verbose" = FALSE)
quanteda_options("threads")
```

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

---

spacyr-methods *Extensions for and from spacy_parse objects*

Description

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

Details

`spacy_parse(x, ...)` and `spacy_tokenize(x, ...)` work directly on **quanteda** corpus objects.

In the methods below, `x` is an object returned by `spacy_parse` (or, for `spacy_parse`, a corpus object), and `...` is not used.

`docnames(x)` returns the document names

`ndoc(x)` returns the number of documents

`ntoken(x, ...)` returns the number of tokens by document

`ntype(x, ...)` returns the number of types (unique tokens) by document

`nsentence(x)` returns the number of sentences by document

Examples

```
## Not run:
library("spacyr")
spacy_initialize()

corp <- corpus(c(doc1 = "And now, now, now for something completely different.",
                 doc2 = "Jack and Jill are children."))
spacy_tokenize(corp)
(parsed <- spacy_parse(corp))

ntype(parsed)
ntoken(parsed)
ndoc(parsed)
docnames(parsed)

## End(Not run)
```

sparsity *Compute the sparsity of a document-feature matrix*

Description

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

Usage

```
sparsity(x)
```

Arguments

x	the document-feature matrix

Examples

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

textmodels *Models for scaling and classification of textual data*

Description

The `textmodel_*()` functions formerly in **quanteda** have now been moved to the **quanteda.textmodels** package.

See Also

`quanteda.textmodels::quanteda.textmodels-package`

textplots *Plots for textual data*

Description

The `textplot_*()` functions formerly in **quanteda** have now been moved to the **quanteda.textplots** package.

See Also

`quanteda.textplots::quanteda.textplots-package`

textstats *Statistics for textual data*

Description

The `textstat_*()` functions formerly in **quanteda** have now been moved to the **quanteda.textstats** package.

See Also

`quanteda.textstats::quanteda.textstats-package`

tokens *Construct a tokens object*

Description

Construct a tokens object, either by importing a named list of characters from an external tokenizer, or by calling the internal **quanteda** tokenizer.

`tokens()` can also be applied to tokens class objects, which means that the removal rules can be applied post-tokenization, although it should be noted that it will not be possible to remove things that are not present. For instance, if the `tokens` object has already had punctuation removed, then `tokens(x, remove_punct = TRUE)` will have no additional effect.

Usage

```
tokens(
  x,
  what = "word",
  remove_punct = FALSE,
  remove_symbols = FALSE,
  remove_numbers = FALSE,
  remove_url = FALSE,
  remove_separators = TRUE,
  split_hyphens = FALSE,
  split_tags = FALSE,
  normalize = FALSE,
  include_docvars = TRUE,
  padding = FALSE,
  concatenator = "_",
  verbose = quanteda_options("verbose"),
  ...,
  xptr = FALSE
)
```

Arguments

x	the input object to the tokens constructor; a tokens, corpus or character object to tokenize.
what	character; which tokenizer to use. The default what = "word" is the current version of the **quanteda** tokenizer, set by quanteda_options(tokens_tokenizer_word). Legacy tokenizers (version < 2) are also supported, including the default what = "word4". See the Details and quanteda Tokenizers below.
remove_punct	logical; if TRUE remove all characters in the Unicode "Punctuation" [P] class, with exceptions for those used as prefixes for valid social media tags if preserve_tags = TRUE
remove_symbols	logical; if TRUE remove all characters in the Unicode "Symbol" [S] class
remove_numbers	logical; if TRUE remove tokens that consist only of numbers, but not words that start with digits, e.g. 2day
remove_url	logical; if TRUE removes URLs (http, https, ftp, sftp) and email addresses.
remove_separators	logical; if TRUE remove separators and separator characters (Unicode "Separator" [Z] and "Control" [C] categories)
split_hyphens	logical; if FALSE, do not split words that are connected by hyphenation and hyphenation-like characters in between words, e.g. "self-aware" becomes c("self", "-", "aware")
split_tags	logical; if FALSE, do not split social media tags defined in quanteda_options(). The default patterns are pattern_hashtag = "#\\w+#?" and pattern_username = "@[a-zA-Z0-9_]+".
normalize	logical; if TRUE, Unicode quotation marks and hyphens are replaced by their ASCII equivalent. See details.
include_docvars	if TRUE, pass docvars through to the tokens object. Does not apply when the input is a character data or a list of characters.
padding	if TRUE, leave an empty string where the removed tokens previously existed. This is useful if a positional match is needed between the pre- and post-selected tokens, for instance if a window of adjacency needs to be computed.
concatenator	character; the concatenation character that will connect the tokens making up a multi-token sequence. See details.
verbose	if TRUE, print timing messages to the console.
...	used to pass arguments among the functions.
xptr	if TRUE, returns a tokens_xptr class object.

Details

As of version 2, the choice of tokenizer is left more to the user, and tokens() is treated more as a constructor (from a named list) than a tokenizer. This allows users to use any other tokenizer that returns a named list, and to use this as an input to tokens(), with removal and splitting rules applied after this has been constructed (passed as arguments). These removal and splitting rules are conservative and will not remove or split anything, however, unless the user requests it.

You usually do not want to split hyphenated words or social media tags, but extra steps required to preserve such special tokens. If there are many random characters in your texts, you should `split_hyphens = TRUE` and `split_tags = TRUE` to avoid a slowdown in tokenization.

To construct a tokens object from a list with no additional processing, call `as.tokens()` instead of `tokens()`.

normalize:

If `normalize = TRUE`, Unicode characters are replaced by their ASCII equivalent to make pattern matching (e.g. stop words) easier: `[\u201C\u201D\u201F]` to the double quotation; `[\u2018\u201B\u2019]` to the single quotation mark; and `\u002D\u2010\u2011\u2012\u2013\u2014\u2015]` to a hyphen.

concatenator:

The choice of `concatenator` does not change tokenization immediately but affects operations in the downstream. Tokens sequences are concatenated with the `concatenator` by tokens_compound into a single tokens. It is possible to change the `concatenator` by reconstructing the object using `tokens()`, but it should be done with caution when the `concatenator` is an empty character "" (typically in Chinese and Japanese) because the old concatenator cannot be replaced with the new one.

tokenizers:

Using external tokenizers is best done by piping the output from these other tokenizers into the `tokens()` constructor, with additional removal and splitting options applied at the construction stage. These will only have an effect, however, if the tokens exist for which removal is specified at in the `tokens()` call. For instance, it is impossible to remove punctuation if the input list to `tokens()` already had its punctuation tokens removed at the external tokenization stage.

Recommended tokenizers are those from the **tokenizers** package, which are generally faster than the default (built-in) tokenizer but always splits infix hyphens, or **spacyr**. The default tokenizer in **quanteda** is very smart, however, and if you do not have special requirements, it works extremely well for most languages as well as text from social media (including hashtags and usernames).

The default word tokenizer `what = "word"` is updated in major version 4. It is even smarter than the v2 and v3 versions, with additional options for customization. See `tokenize_word4()` for full details.

The default tokenizer splits tokens using stri_split_boundaries(x, type = "word") but by default preserves infix hyphens (e.g. "self-funding"), URLs, and social media "tag" characters (#hashtags and @usernames), and email addresses. The rules defining a valid "tag" can be found at https://www.hashtags.org/featured/what-characters-can-a-hashtag-include/ for hashtags and at https://help.twitter.com/en/managing-your-account/twitter-username-rules for usernames.

For backward compatibility, the following older tokenizers are also supported through `what`:

`"word1"` (legacy) implements similar behaviour to the version of `what = "word"` found in pre-version 2. (It preserves social media tags and infix hyphens, but splits URLs.) "word1" is also slower than "word2" and "word4". In "word1", the argument `remove_twitter` controlled whether social media tags were preserved or removed, even when `remove_punct = TRUE`. This argument is not longer functional in versions >= 2, but equivalent control can be had using the `split_tags` argument and selective tokens removals.

`"word2", "word3"` (legacy) implements similar behaviour to the versions of "word" found in **quanteda** versions 3 and 4.

`"fasterword"` (legacy) splits on whitespace and control characters, using `stringi::stri_split_charclass(x, "[\\p{Z}\\p{C}]+")`

`"fastestword"` (legacy) splits on the space character, using `stringi::stri_split_fixed(x, " ")`

`"character"` tokenization into individual characters

`"sentence"` sentence segmenter based on stri_split_boundaries, but with additional rules to avoid splits on words like "Mr." that would otherwise incorrectly be detected as sentence boundaries. For better sentence tokenization, consider using **spacyr**.

Value

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

See Also

`tokens_ngrams()`, `tokens_skipgrams()`, `tokens_compound()`, `tokens_lookup()`, `concat()`, `as.list.tokens()`, `as.tokens()`

Examples

```
txt <- c(doc1 = "A sentence, showing how tokens() works.",
         doc2 = "@quantedainit and #textanalysis https://example.com?p=123.",
         doc3 = "Self-documenting code??",
         doc4 = "£1,000,000 for 50¢ is gr8 4ever \U0001f600")
tokens(txt)
tokens(txt, what = "word1")

# removing punctuation marks but keeping tags and URLs
tokens(txt[1:2], remove_punct = TRUE)

# splitting hyphenated words
tokens(txt[3])
tokens(txt[3], split_hyphens = TRUE)

# symbols and numbers
tokens(txt[4])
tokens(txt[4], remove_numbers = TRUE)
tokens(txt[4], remove_numbers = TRUE, remove_symbols = TRUE)

## Not run: # using other tokenizers
tokens(tokenizers::tokenize_words(txt[4]), remove_symbols = TRUE)
tokenizers::tokenize_words(txt, lowercase = FALSE, strip_punct = FALSE) |>
    tokens(remove_symbols = TRUE)
tokenizers::tokenize_characters(txt[3], strip_non_alphanum = FALSE) |>
    tokens(remove_punct = TRUE)
tokenizers::tokenize_sentences(
    "The quick brown fox.  It jumped over the lazy dog.") |>
    tokens()

## End(Not run)
```

tokens_annotate *Annotate a tokens object using a dictionary*

Description

Insert dictionary keys as tags in a tokens object where the dictionary patterns are found.

Usage

```
tokens_annotate(
  x,
  dictionary,
  levels = 1:5,
  valuetype = c("glob", "regex", "fixed"),
  case_insensitive = TRUE,
  marker = "#",
  capkeys = TRUE,
  nested_scope = c("key", "dictionary"),
  apply_if = NULL,
  verbose = quanteda_options("verbose")
)
```

Arguments

x	the tokens object to which the dictionary will be applied
dictionary	the dictionary-class object that will be applied to x
levels	integers specifying the levels of entries in a hierarchical dictionary that will be applied. The top level is 1, and subsequent levels describe lower nesting levels. Values may be combined, even if these levels are not contiguous, e.g. levels = c(1:3) will collapse the second level into the first, but record the third level (if present) collapsed below the first (see examples).
valuetype	the type of pattern matching: "glob" for "glob"-style wildcard expressions; "regex" for regular expressions; or "fixed" for exact matching. See valuetype for details.
case_insensitive	logical; if TRUE, ignore case when matching a pattern or dictionary values
marker	characters that are added before and after the dictionary keys to create tags.
capkeys	if TRUE, convert dictionary keys to uppercase to distinguish them from unmatched tokens.
nested_scope	how to treat matches from different dictionary keys that are nested. When one value is nested within another, such as "a b" being nested within "a b c", then tokens_lookup() will match the longer. When nested_scope = "key", this longer-match priority is applied only within the key, while "dictionary" applies it across keys, matching only the key with the longer pattern, not the matches nested within that longer pattern from other keys. See Details.

apply_if	logical vector of length ndoc(x); documents are modified only when corresponding values are TRUE, others are left unchanged.
verbose	if TRUE print a summary of the object returned by the function: the number of documents and, depending on the object, the number of characters, tokens and types, or features and occurrences. The number of tokens includes paddings, but the number of features excludes them.

See Also

tokens_lookup

Examples

```
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_annotate(toks, dict)
```

---

tokens_chunk *Segment tokens object by chunks of a given size*

Description

Segment tokens into new documents of equally sized token lengths, with the possibility of overlapping the chunks.

Usage

```
tokens_chunk(
  x,
  size,
  overlap = 0,
  use_docvars = TRUE,
  verbose = quanteda_options("verbose")
)
```

Arguments

x	tokens object whose token elements will be segmented into chunks
size	integer; the token length of the chunks
overlap	integer; the number of tokens in a chunk to be taken from the last overlap tokens from the preceding chunk

use_docvars	if TRUE, repeat the docvar values for each chunk; if FALSE, drop the docvars in the chunked tokens
verbose	if TRUE print a summary of the object returned by the function: the number of documents and, depending on the object, the number of characters, tokens and types, or features and occurrences. The number of tokens includes paddings, but the number of features excludes them.

Value

A tokens object whose documents have been split into chunks of length size.

See Also

tokens_segment()

Examples

```
txts <- c(doc1 = "Fellow citizens, I am again called upon by the voice of
                  my country to execute the functions of its Chief Magistrate.",
          doc2 = "When the occasion proper for it shall arrive, I shall
                  endeavor to express the high sense I entertain of this
                  distinguished honor.")
toks <- tokens(txts)
tokens_chunk(toks, size = 5)
tokens_chunk(toks, size = 5, overlap = 4)
```

---

tokens_compound *Convert token sequences into compound tokens*

Description

Replace multi-token sequences with a multi-word, or "compound" token. The resulting compound tokens will represent a phrase or multi-word expression, concatenated with concatenator (by default, the "_" character) to form a single "token". This ensures that the sequences will be processed subsequently as single tokens, for instance in constructing a dfm.

Usage

```
tokens_compound(
  x,
  pattern,
  valuetype = c("glob", "regex", "fixed"),
  concatenator = concat(x),
  window = 0L,
  case_insensitive = TRUE,
  join = TRUE,
  keep_unigrams = FALSE,
  apply_if = NULL,
  verbose = quanteda_options("verbose")
)
```

Arguments

x	an input tokens object
pattern	a character vector, list of character vectors, dictionary, or collocations object. See pattern for details.
valuetype	the type of pattern matching: "glob" for "glob"-style wildcard expressions; "regex" for regular expressions; or "fixed" for exact matching. See valuetype for details.
concatenator	character; the concatenation character that will connect the tokens making up a multi-token sequence.
window	integer; a vector of length 1 or 2 that specifies size of the window of tokens adjacent to pattern that will be compounded with matches to pattern. The window can be asymmetric if two elements are specified, with the first giving the window size before pattern and the second the window size after. If paddings (empty "" tokens) are found, window will be shrunk to exclude them.
case_insensitive	logical; if TRUE, ignore case when matching a pattern or dictionary values
join	logical; if TRUE, join overlapping compounds into a single compound; otherwise, form these separately. See examples.
keep_unigrams	if TRUE, keep the original tokens.
apply_if	logical vector of length ndoc(x); documents are modified only when corresponding values are TRUE, others are left unchanged.
verbose	if TRUE print a summary of the object returned by the function: the number of documents and, depending on the object, the number of characters, tokens and types, or features and occurrences. The number of tokens includes paddings, but the number of features excludes them.

Value

A tokens object in which the token sequences matching pattern have been replaced by new compounded "tokens" joined by the concatenator.

Note

Patterns to be compounded (naturally) consist of multi-word sequences, and how these are expected in pattern is very specific. If the elements to be compounded are supplied as space-delimited elements of a character vector, wrap the vector in phrase(). If the elements to be compounded are separate elements of a character vector, supply it as a list where each list element is the sequence of character elements.

See the examples below.

Examples

```
txt <- "The United Kingdom is leaving the European Union."
toks <- tokens(txt, remove_punct = TRUE)

# character vector - not compounded
```

```
tokens_compound(toks, c("United", "Kingdom", "European", "Union"))

# elements separated by spaces - not compounded
tokens_compound(toks, c("United Kingdom", "European Union"))

# list of characters - is compounded
tokens_compound(toks, list(c("United", "Kingdom"), c("European", "Union")))

# elements separated by spaces, wrapped in phrase() - is compounded
tokens_compound(toks, phrase(c("United Kingdom", "European Union")))

# supplied as values in a dictionary (same as list) - is compounded
# (keys do not matter)
tokens_compound(toks, dictionary(list(key1 = "United Kingdom",
                                      key2 = "European Union")))
# pattern as dictionaries with glob matches
tokens_compound(toks, dictionary(list(key1 = c("U* K*"))), valuetype = "glob")

# note the differences caused by join = FALSE
compounds <- list(c("the", "European"), c("European", "Union"))
tokens_compound(toks, pattern = compounds, join = TRUE)
tokens_compound(toks, pattern = compounds, join = FALSE)

# use window to form ngrams
tokens_remove(toks, pattern = stopwords("en")) |>
    tokens_compound(pattern = "leav*", join = FALSE, window = c(0, 3))
```

---

tokens_group *Combine documents in a tokens object by a grouping variable*

Description

Combine documents in a tokens object by a grouping variable, by concatenating the tokens in the order of the documents within each grouping variable.

Usage

```
tokens_group(
  x,
  groups = docid(x),
  fill = FALSE,
  env = NULL,
  verbose = quanteda_options("verbose")
)
```

Arguments

x	tokens object

groups — grouping variable for sampling, equal in length to the number of documents. This will be evaluated in the docvars data.frame, so that docvars may be referred to by name without quoting. This also changes previous behaviours for groups. See news(Version >= "3.0", package = "quanteda") for details.

fill — logical; if TRUE and groups is a factor, then use all levels of the factor when forming the new documents of the grouped object. This will result in a new "document" with empty content for levels not observed, but for which an empty document may be needed. If groups is a factor of dates, for instance, then fill = TRUE ensures that the new object will consist of one new "document" by date, regardless of whether any documents previously existed with that date. Has no effect if the groups variable(s) are not factors.

env — an environment or a list object in which x is searched. Passed to substitute for non-standard evaluation.

verbose — if TRUE print a summary of the object returned by the function: the number of documents and, depending on the object, the number of characters, tokens and types, or features and occurrences. The number of tokens includes paddings, but the number of features excludes them.

Value

a tokens object whose documents are equal to the unique group combinations, and whose tokens are the concatenations of the tokens by group. Document-level variables that have no variation within groups are saved in docvars. Document-level variables that are lists are dropped from grouping, even when these exhibit no variation within groups.

Examples

```
corp <- corpus(c("a a b", "a b c c", "a c d d", "a c c d"),
               docvars = data.frame(grp = c("grp1", "grp1", "grp2", "grp2")))
toks <- tokens(corp)
tokens_group(toks, groups = grp)
tokens_group(toks, groups = c(1, 1, 2, 2))

# with fill
tokens_group(toks, groups = factor(c(1, 1, 2, 2), levels = 1:3))
tokens_group(toks, groups = factor(c(1, 1, 2, 2), levels = 1:3), fill = TRUE)
```

---

tokens_lookup *Apply a dictionary to a tokens object*

Description

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

Usage

```
tokens_lookup(
  x,
  dictionary,
  levels = 1:5,
  valuetype = c("glob", "regex", "fixed"),
  case_insensitive = TRUE,
  capkeys = !exclusive,
  exclusive = TRUE,
  nomatch = NULL,
  append_key = FALSE,
  separator = "/",
  concatenator = concat(x),
  nested_scope = c("key", "dictionary"),
  apply_if = NULL,
  verbose = quanteda_options("verbose")
)
```

Arguments

x	the tokens object to which the dictionary will be applied
dictionary	the dictionary-class object that will be applied to x
levels	integers specifying the levels of entries in a hierarchical dictionary that will be applied. The top level is 1, and subsequent levels describe lower nesting levels. Values may be combined, even if these levels are not contiguous, e.g. levels = c(1:3) will collapse the second level into the first, but record the third level (if present) collapsed below the first (see examples).
valuetype	the type of pattern matching: "glob" for "glob"-style wildcard expressions; "regex" for regular expressions; or "fixed" for exact matching. See valuetype for details.
case_insensitive	logical; if TRUE, ignore case when matching a pattern or dictionary values
capkeys	if TRUE, convert dictionary keys to uppercase to distinguish them from unmatched tokens.
exclusive	if TRUE, remove all features not in dictionary, otherwise, replace values in dictionary with keys while leaving other features unaffected.
nomatch	an optional character naming a new key for tokens that do not matched to a dictionary values If NULL (default), do not record unmatched tokens.
append_key	if TRUE, annotate matched tokens with keys.
separator	a character to separate tokens and keys when append_key = TRUE.
concatenator	the concatenation character that will connect the words making up the multi-word sequences.
nested_scope	how to treat matches from different dictionary keys that are nested. When one value is nested within another, such as "a b" being nested within "a b c", then tokens_lookup() will match the longer. When nested_scope = "key",

this longer-match priority is applied only within the key, while "dictionary" applies it across keys, matching only the key with the longer pattern, not the matches nested within that longer pattern from other keys. See Details.

apply_if — logical vector of length ndoc(x); documents are modified only when corresponding values are TRUE, others are left unchanged.

verbose — if TRUE print a summary of the object returned by the function: the number of documents and, depending on the object, the number of characters, tokens and types, or features and occurrences. The number of tokens includes paddings, but the number of features excludes them.

Details

Dictionary values may consist of sequences, and there are different methods of counting key matches based on values that are nested or that overlap.

When two different keys in a dictionary are nested matches of one another, the nested_scope options provide the choice of matching each key's values independently (the "key") option, or just counting the longest match (the "dictionary" option). Values that are nested *within* the same key are always counted as a single match. See the last example below comparing the *New York* and *New York Times* for these two different behaviours.

Overlapping values, such as "a b" and "b a" are currently always considered as separate matches if they are in different keys, or as one match if the overlap is within the same key.

Note: apply_if This applies the dictionary lookup only to documents that match the logical condition. When exclusive = TRUE (the default), however, this means that empty documents will be returned for those not meeting the condition, since no lookup will be applied and hence no tokens replaced by matching keys.

See Also

tokens_replace

Examples

```
toks1 <- tokens(data_corpus_inaugural)
dict1 <- dictionary(list(country = "united states",
                  law=c("law*", "constitution"),
                  freedom=c("free*", "libert*")))
dfm(tokens_lookup(toks1, dict1, valuetype = "glob", verbose = TRUE))
dfm(tokens_lookup(toks1, dict1, valuetype = "glob", verbose = TRUE, nomatch = "NONE"))

dict2 <- dictionary(list(country = "united states",
                      law = c("law", "constitution"),
                      freedom = c("freedom", "liberty")))
# dfm(applyDictionary(toks1, dict2, valuetype = "fixed"))
dfm(tokens_lookup(toks1, dict2, valuetype = "fixed"))

# hierarchical dictionary example
txt <- c(d1 = "The United States has the Atlantic Ocean and the Pacific Ocean.",
         d2 = "Britain and Ireland have the Irish Sea and the English Channel.")
toks2 <- tokens(txt)
```

```
dict3 <- dictionary(list(US = list(Countries = c("States"),
                                   oceans = c("Atlantic", "Pacific")),
                         Europe = list(Countries = c("Britain", "Ireland"),
                                       oceans = list(west = "Irish Sea",
                                                     east = "English Channel"))))
tokens_lookup(toks2, dict3, levels = 1)
tokens_lookup(toks2, dict3, levels = 2)
tokens_lookup(toks2, dict3, levels = 1:2)
tokens_lookup(toks2, dict3, levels = 3)
tokens_lookup(toks2, dict3, levels = c(1,3))
tokens_lookup(toks2, dict3, levels = c(2,3))

# show unmatched tokens
tokens_lookup(toks2, dict3, nomatch = "_UNMATCHED")

# nested matching differences
dict4 <- dictionary(list(paper = "New York Times", city = "New York"))
toks4 <- tokens("The New York Times is a New York paper.")
tokens_lookup(toks4, dict4, nested_scope = "key", exclusive = FALSE)
tokens_lookup(toks4, dict4, nested_scope = "dictionary", exclusive = FALSE)
```

`tokens_match` *Match the tokens IDs with given types*

Description

Match the token types and IDs of multiple tokens objects using a character vector.

Usage

```
tokens_match(x, types, verbose = quanteda_options("verbose"))
```

Arguments

`x`	the tokens object.
`types`	character vector for the token types to be matched in the resulting tokens. Tokens not included in `types` are all removed.
`verbose`	if `TRUE` print a summary of the object returned by the function: the number of documents and, depending on the object, the number of characters, tokens and types, or features and occurrences. The number of tokens includes paddings, but the number of features excludes them.

See Also

dfm_match

Examples

```
txt <- c("This is text one", "The text two", "This is text three")
(toks1 <- tokens(txt[1:2]))
(toks2 <- tokens(txt[2:3]))
(toks3 <- tokens_match(toks1, types(toks2)))
identical(types(toks2), types(toks3))
```

---

tokens_ngrams *Create n-grams and skip-grams from tokens*

Description

Create a set of n-grams (tokens in sequence) from already tokenized text objects, with an optional skip argument to form skip-grams. Both the n-gram 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 = concat(x),
  apply_if = NULL,
  verbose = quanteda_options("verbose")
)

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

tokens_skipgrams(
  x,
  n,
  skip,
  concatenator = concat(x),
  apply_if = NULL,
  verbose = quanteda_options("verbose")
)
```

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 n-gram. 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 n-grams, 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
apply_if	logical vector of length ndoc(x); documents are modified only when corresponding values are TRUE, others are left unchanged.
verbose	if TRUE print a summary of the object returned by the function: the number of documents and, depending on the object, the number of characters, tokens and types, or features and occurrences. The number of tokens includes paddings, but the number of features excludes them.

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 n-gram construction on tokenized texts.

tokens_skipgrams() is a wrapper to tokens_ngrams() that requires arguments to be supplied for both n and skip. For k-skip skip-grams, 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 n-gram which is a superset of all n-grams and each $(k - i)$ skip-gram until $(k - i) == 0$ (which includes 0 skip-grams).

Value

a tokens object consisting a list of character vectors of n-grams, 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.

References

Guthrie, David, Ben Allison, Wei Liu, Louise Guthrie, and Yorick Wilks. 2006. "A Closer Look at Skip-Gram Modelling." https://aclanthology.org/L06-1210/

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 = " ")
```

```
# 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_replace` *Replace tokens in a tokens object*

Description

Substitute token types based on vectorized one-to-one matching. Since this function is created for lemmatization or user-defined stemming. It supports substitution of multi-word features by multi-word features, but substitution is fastest when `pattern` and `replacement` are character vectors and `valuetype = "fixed"` as the function only substitute types of tokens. Please use `tokens_lookup()` with `exclusive = FALSE` to replace dictionary values.

Usage

```
tokens_replace(
  x,
  pattern,
  replacement,
  valuetype = "glob",
  case_insensitive = TRUE,
  apply_if = NULL,
  verbose = quanteda_options("verbose")
)
```

Arguments

`x`	tokens object whose token elements will be replaced
`pattern`	a character vector or list of character vectors. See pattern for more details.
`replacement`	a character vector or (if `pattern` is a list) list of character vectors of the same length as `pattern`
`valuetype`	the type of pattern matching: `"glob"` for "glob"-style wildcard expressions; `"regex"` for regular expressions; or `"fixed"` for exact matching. See valuetype for details.
`case_insensitive`	logical; if `TRUE`, ignore case when matching a `pattern` or dictionary values
`apply_if`	logical vector of length `ndoc(x)`; documents are modified only when corresponding values are `TRUE`, others are left unchanged.
`verbose`	if `TRUE` print a summary of the object returned by the function: the number of documents and, depending on the object, the number of characters, tokens and types, or features and occurrences. The number of tokens includes paddings, but the number of features excludes them.

See Also

tokens_lookup

Examples

```
toks1 <- tokens(data_corpus_inaugural, remove_punct = TRUE)

# lemmatization
taxwords <- c("tax", "taxing", "taxed", "taxed", "taxation")
lemma <- rep("TAX", length(taxwords))
toks2 <- tokens_replace(toks1, taxwords, lemma, valuetype = "fixed")
kwic(toks2, "TAX") |>
    tail(10)

# stemming
type <- types(toks1)
stem <- char_wordstem(type, "porter")
toks3 <- tokens_replace(toks1, type, stem, valuetype = "fixed", case_insensitive = FALSE)
identical(toks3, tokens_wordstem(toks1, "porter"))

# multi-multi substitution
toks4 <- tokens_replace(toks1, phrase(c("Supreme Court")),
                        phrase(c("Supreme Court of the United States")))
kwic(toks4, phrase(c("Supreme Court of the United States")))
```

---

tokens_sample *Randomly sample documents from a tokens object*

Description

Take a random sample of documents of the specified size from a corpus, with or without replacement, optionally by grouping variables or with probability weights.

Usage

```
tokens_sample(
  x,
  size = NULL,
  replace = FALSE,
  prob = NULL,
  by = NULL,
  env = NULL,
  verbose = quanteda_options("verbose")
)
```

Arguments

x	a tokens object whose documents will be sampled
size	a positive number, the number of documents to select; when used with by, the number to select from each group or a vector equal in length to the number of groups defining the samples to be chosen in each category of by. By defining a size larger than the number of documents, it is possible to oversample when replace = TRUE.
replace	if TRUE, sample with replacement
prob	a vector of probability weights for obtaining the elements of the vector being sampled. May not be applied when by is used.
by	optional grouping variable for sampling. This will be evaluated in the docvars data.frame, so that docvars may be referred to by name without quoting. This also changes previous behaviours for by. See news(Version >= "2.9", package = "quanteda") for details.
env	an environment or a list object in which x is searched. Passed to substitute for non-standard evaluation.
verbose	if TRUE print a summary of the object returned by the function: the number of documents and, depending on the object, the number of characters, tokens and types, or features and occurrences. The number of tokens includes paddings, but the number of features excludes them.

Value

a tokens object (re)sampled on the documents, containing the document variables for the documents sampled.

See Also

sample

Examples

```
set.seed(123)
toks <- tokens(data_corpus_inaugural[1:6])
toks
tokens_sample(toks)
tokens_sample(toks, replace = TRUE) |> docnames()
tokens_sample(toks, size = 3, replace = TRUE) |> docnames()

# sampling using by
docvars(toks)
tokens_sample(toks, size = 2, replace = TRUE, by = Party) |> docnames()
```

`tokens_segment` *Segment tokens object by patterns*

Description

Segment tokens by splitting on a pattern match. This is useful for breaking the tokenized texts into smaller document units, based on a regular pattern or a user-supplied annotation. While it normally makes more sense to do this at the corpus level (see `corpus_segment()`), `tokens_segment` provides the option to perform this operation on tokens.

Usage

```
tokens_segment(
  x,
  pattern,
  valuetype = c("glob", "regex", "fixed"),
  case_insensitive = TRUE,
  extract_pattern = FALSE,
  pattern_position = c("before", "after"),
  use_docvars = TRUE,
  verbose = quanteda_options("verbose")
)
```

Arguments

`x`	tokens object whose token elements will be segmented
`pattern`	a character vector, list of character vectors, dictionary, or collocations object. See pattern for details.
`valuetype`	the type of pattern matching: `"glob"` for "glob"-style wildcard expressions; `"regex"` for regular expressions; or `"fixed"` for exact matching. See valuetype for details.
`case_insensitive`	logical; if `TRUE`, ignore case when matching a `pattern` or dictionary values
`extract_pattern`	remove matched patterns from the texts and save in docvars, if `TRUE`
`pattern_position`	either `"before"` or `"after"`, depending on whether the pattern precedes the text (as with a tag) or follows the text (as with punctuation delimiters)
`use_docvars`	if `TRUE`, repeat the docvar values for each segmented text; if `FALSE`, drop the docvars in the segmented corpus. Dropping the docvars might be useful in order to conserve space or if these are not desired for the segmented corpus.
`verbose`	if `TRUE` print a summary of the object returned by the function: the number of documents and, depending on the object, the number of characters, tokens and types, or features and occurrences. The number of tokens includes paddings, but the number of features excludes them.

Value

tokens_segment returns a tokens object whose documents have been split by patterns

Examples

```
txts <- "Fellow citizens, I am again called upon by the voice of my country to
execute the functions of its Chief Magistrate. When the occasion proper for
it shall arrive, I shall endeavor to express the high sense I entertain of
this distinguished honor."
toks <- tokens(txts)

# split by any punctuation
tokens_segment(toks, "^\\p{Sterm}$", valuetype = "regex",
               extract_pattern = TRUE,
               pattern_position = "after")
tokens_segment(toks, c(".", "?", "!"), valuetype = "fixed",
               extract_pattern = TRUE,
               pattern_position = "after")
```

---

tokens_select *Select or remove tokens from a tokens object*

Description

These function select or discard tokens from a tokens object. For convenience, the functions tokens_remove and tokens_keep are defined as shortcuts for tokens_select(x, pattern, selection = "remove") and tokens_select(x, pattern, selection = "keep"), respectively. The most common usage for tokens_remove will be to eliminate stop words from a text or text-based object, while the most common use of tokens_select will be to select tokens with only positive pattern matches from a list of regular expressions, including a dictionary. startpos and endpos determine the positions of tokens searched for pattern and areas affected are expanded by window.

Usage

```
tokens_select(
  x,
  pattern,
  selection = c("keep", "remove"),
  valuetype = c("glob", "regex", "fixed"),
  case_insensitive = TRUE,
  padding = FALSE,
  window = 0,
  min_nchar = NULL,
  max_nchar = NULL,
  startpos = 1L,
  endpos = -1L,
  apply_if = NULL,
  verbose = quanteda_options("verbose")
```

```
)

tokens_remove(x, ...)

tokens_keep(x, ...)
```

Arguments

x	tokens object whose token elements will be removed or kept
pattern	a character vector, list of character vectors, dictionary, or collocations object. See pattern for details.
selection	whether to "keep" or "remove" the tokens matching pattern
valuetype	the type of pattern matching: "glob" for "glob"-style wildcard expressions; "regex" for regular expressions; or "fixed" for exact matching. See valuetype for details.
case_insensitive	logical; if TRUE, ignore case when matching a pattern or dictionary values
padding	if TRUE, leave an empty string where the removed tokens previously existed. This is useful if a positional match is needed between the pre- and post-selected tokens, for instance if a window of adjacency needs to be computed.
window	integer of length 1 or 2; the size of the window of tokens adjacent to pattern that will be selected. The window is symmetric unless a vector of two elements is supplied, in which case the first element will be the token length of the window before pattern, and the second will be the token length of the window after pattern. The default is 0, meaning that only the pattern matched token(s) are selected, with no adjacent terms. Terms from overlapping windows are never double-counted, but simply returned in the pattern match. This is because tokens_select never redefines the document units; for this, see kwic().
min_nchar, max_nchar	optional numerics specifying the minimum and maximum length in characters for tokens to be removed or kept; defaults are NULL for no limits. These are applied after (and hence, in addition to) any selection based on pattern matches.
startpos, endpos	integer; position of tokens in documents where pattern matching starts and ends, where 1 is the first token in a document. For negative indexes, counting starts at the ending token of the document, so that -1 denotes the last token in the document, -2 the second to last, etc. When the length of the vector is equal to ndoc, tokens in corresponding positions will be selected; when it is less than ndoc, values are repeated to make them equal in length.
apply_if	logical vector of length ndoc(x); documents are modified only when corresponding values are TRUE, others are left unchanged.
verbose	if TRUE print a summary of the object returned by the function: the number of documents and, depending on the object, the number of characters, tokens and types, or features and occurrences. The number of tokens includes paddings, but the number of features excludes them.

... additional arguments passed by tokens_remove and tokens_keep to tokens_select. Cannot include selection.

Value

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

Examples

```
## tokens_select with simple examples
toks <- as.tokens(list(letters, LETTERS))
tokens_select(toks, c("b", "e", "f"), selection = "keep", padding = FALSE)
tokens_select(toks, c("b", "e", "f"), selection = "keep", padding = TRUE)
tokens_select(toks, c("b", "e", "f"), selection = "remove", padding = FALSE)
tokens_select(toks, c("b", "e", "f"), selection = "remove", padding = TRUE)

# how case_insensitive works
tokens_select(toks, c("b", "e", "f"), selection = "remove", case_insensitive = TRUE)
tokens_select(toks, c("b", "e", "f"), selection = "remove", case_insensitive = FALSE)

# use window
tokens_select(toks, c("b", "f"), selection = "keep", window = 1)
tokens_select(toks, c("b", "f"), selection = "remove", window = 1)
tokens_remove(toks, c("b", "f"), window = c(0, 1))
tokens_select(toks, pattern = c("e", "g"), window = c(1, 2))

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

# token_keep example: keep two-letter words
tokens_keep(tokens(txt, remove_punct = TRUE), "??")
```

---

tokens_split *Split tokens by a separator pattern*

Description

Replaces tokens by multiple replacements consisting of elements split by a separator pattern, with the option of retaining the separator. This function effectively reverses the operation of tokens_compound().

Usage

```
tokens_split(
  x,
  separator = " ",
  valuetype = c("fixed", "regex"),
  remove_separator = TRUE,
  apply_if = NULL,
  verbose = quanteda_options("verbose")
)
```

Arguments

x	a tokens object
separator	a single-character pattern match by which tokens are separated
valuetype	the type of pattern matching: "glob" for "glob"-style wildcard expressions; "regex" for regular expressions; or "fixed" for exact matching. See valuetype for details.
remove_separator	if TRUE, remove separator from new tokens
apply_if	logical vector of length ndoc(x); documents are modified only when corresponding values are TRUE, others are left unchanged.
verbose	if TRUE print a summary of the object returned by the function: the number of documents and, depending on the object, the number of characters, tokens and types, or features and occurrences. The number of tokens includes paddings, but the number of features excludes them.

Examples

```
# undo tokens_compound()
toks1 <- tokens("pork barrel is an idiomatic multi-word expression")
tokens_compound(toks1, phrase("pork barrel"))
tokens_compound(toks1, phrase("pork barrel")) |>
    tokens_split(separator = "_")

# similar to tokens(x, remove_hyphen = TRUE) but post-tokenization
toks2 <- tokens("UK-EU negotiation is not going anywhere as of 2018-12-24.")
tokens_split(toks2, separator = "-", remove_separator = FALSE)
```

---

tokens_subset *Extract a subset of a tokens*

Description

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

Usage

```
tokens_subset(
  x,
  subset,
  min_ntoken = NULL,
  max_ntoken = NULL,
  drop_docid = TRUE,
  verbose = quanteda_options("verbose"),
  ...
)
```

Arguments

x	tokens object to be subsetted.
subset	logical expression indicating the documents to keep: missing values are taken as false.
min_ntoken, max_ntoken	minimum and maximum lengths of the documents to extract.
drop_docid	if TRUE, docid for documents are removed as the result of subsetting.
verbose	if TRUE print a summary of the object returned by the function: the number of documents and, depending on the object, the number of characters, tokens and types, or features and occurrences. The number of tokens includes paddings, but the number of features excludes them.
...	not used

Value

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

See Also

subset.data.frame()

Examples

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

tokens_tolower *Convert the case of tokens*

Description

tokens_tolower() and tokens_toupper() convert the features of a tokens object and re-index the types.

Usage

```
tokens_tolower(x, keep_acronyms = FALSE, verbose = quanteda_options("verbose"))

tokens_toupper(x, verbose = quanteda_options("verbose"))
```

Arguments

x	the input object whose character/tokens/feature elements will be case-converted
keep_acronyms	logical; if TRUE, do not lowercase any all-uppercase words (applies only to *_tolower() functions)
verbose	if TRUE print a summary of the object returned by the function: the number of documents and, depending on the object, the number of characters, tokens and types, or features and occurrences. The number of tokens includes paddings, but the number of features excludes them.

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_trim *Trim tokens using frequency threshold-based feature selection*

Description

Returns a tokens object reduced in size based on document and term frequency, usually in terms of a minimum frequency, but may also be in terms of maximum frequencies. Setting a combination of minimum and maximum frequencies will select features based on a range.

Usage

```
tokens_trim(
  x,
  min_termfreq = NULL,
  max_termfreq = NULL,
  termfreq_type = c("count", "prop", "rank", "quantile"),
  min_docfreq = NULL,
  max_docfreq = NULL,
  docfreq_type = c("count", "prop", "rank", "quantile"),
  max_n = NULL,
  padding = FALSE,
  verbose = quanteda_options("verbose")
)
```

Arguments

x	a dfm object
min_termfreq, max_termfreq	minimum/maximum values of feature frequencies across all documents, below/above which features will be removed
termfreq_type	how min_termfreq and max_termfreq are interpreted. "count" sums the frequencies; "prop" divides the term frequencies by the total sum; "rank" is matched against the inverted ranking of features in terms of overall frequency, so that 1, 2, ... are the highest and second highest frequency features, and so on; "quantile" sets the cutoffs according to the quantiles (see quantile()) of term frequencies.
min_docfreq, max_docfreq	minimum/maximum values of a feature's document frequency, below/above which features will be removed
docfreq_type	specify how min_docfreq and max_docfreq are interpreted. "count" is the same as [docfreq](x, scheme = "count"); "prop" divides the document frequencies by the total sum; "rank" is matched against the inverted ranking of document frequency, so that 1, 2, ... are the features with the highest and second highest document frequencies, and so on; "quantile" sets the cutoffs according to the quantiles (see quantile()) of document frequencies.
max_n	the maximum number of features. Features are selected based on their frequencies, but the earlier elements in featnames or types are chosen when the frequencies are equal.
padding	if TRUE, leave an empty string where the removed tokens previously existed.
verbose	if TRUE print a summary of the object returned by the function: the number of documents and, depending on the object, the number of characters, tokens and types, or features and occurrences. The number of tokens includes paddings, but the number of features excludes them.

Value

A tokens object with reduced size.

See Also

dfm_trim()

Examples

```
toks <- tokens(data_corpus_inaugural)

# keep only words occurring >= 10 times and in >= 2 documents
tokens_trim(toks, min_termfreq = 10, min_docfreq = 2, padding = TRUE)

# keep only words occurring >= 10 times and no more than 90% of the documents
tokens_trim(toks, min_termfreq = 10, max_docfreq = 0.9, docfreq_type = "prop",
            padding = TRUE)
```

---

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 = quanteda_options("language_stemmer"),
  verbose = quanteda_options("verbose")
)

char_wordstem(
  x,
  language = quanteda_options("language_stemmer"),
  check_whitespace = TRUE
)

dfm_wordstem(
  x,
  language = quanteda_options("language_stemmer"),
  verbose = quanteda_options("verbose")
)
```

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)
verbose	if TRUE print a summary of the object returned by the function: the number of documents and, depending on the object, the number of characters, tokens and types, or features and occurrences. The number of tokens includes paddings, but the number of features excludes them.
check_whitespace	logical; if TRUE, stop with a warning when trying to stem inputs containing whitespace

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

https://snowballstem.org/

https://www.iso.org/iso-639-language-code 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(tokens(txt)))
dfm_wordstem(origdfm)
```

tokens_xptr *Methods for tokens_xptr objects*

Description

Methods for creating and testing for `tokens_xptr` objects, which are tokens objects containing pointers to memory locations that can be passed by reference for efficient processing in `tokens_*()` functions that modify them, or for constructing a document-feature matrix without requiring a deep copy to be passed to `dfm()`.

`is.tokens_xptr()` tests whether an object is of class `tokens_xtpr`.

`as.tokens_xptr()` coerces a `tokens` object to an external pointer-based tokens object, or returns a deep copy of a `tokens_xtpr` when `x` is already a `tokens_xtpr` object.

Usage

```
is.tokens_xptr(x)

as.tokens_xptr(x)

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

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

Arguments

`x`	a tokens object to convert or a `tokens_xptr` class object to deep copy.

Value

`is.tokens_xptr()` returns `TRUE` if the object is a external pointer-based tokens object, `FALSE` otherwise.

`as.tokens_xptr()` returns a (deep copy of a) `tokens_xtpr` class object.

topfeatures *Identify the most frequent features in a dfm*

Description

List the most (or least) frequently occurring features in a dfm, either as a whole or separated by document.

Usage

```
topfeatures(
  x,
  n = 10,
  decreasing = TRUE,
  scheme = c("count", "docfreq"),
  groups = NULL
)
```

Arguments

x	the object whose features will be returned
n	how many top features should be returned
decreasing	If TRUE, return the n most frequent features; otherwise return the n least frequent features
scheme	one of count for total feature frequency (within group if applicable), or docfreq for the document frequencies of features
groups	grouping variable for sampling, equal in length to the number of documents. This will be evaluated in the docvars data.frame, so that docvars may be referred to by name without quoting. This also changes previous behaviours for groups. See news(Version >= "3.0", package = "quanteda") for details.

Value

A named numeric vector of feature counts, where the names are the feature labels, or a list of these if groups is given.

Examples

```
dfmat1 <- corpus_subset(data_corpus_inaugural, Year > 1980) |>
    tokens(remove_punct = TRUE) |>
    dfm()
dfmat2 <- dfm_remove(dfmat1, stopwords("en"))

# most frequent features
topfeatures(dfmat1)
topfeatures(dfmat2)

# least frequent features
topfeatures(dfmat2, decreasing = FALSE)

# top features of individual documents
topfeatures(dfmat2, n = 5, groups = docnames(dfmat2))

# grouping by president last name
topfeatures(dfmat2, n = 5, groups = President)

# features by document frequencies
tail(topfeatures(dfmat1, scheme = "docfreq", n = 200))
```

types *Get word types from a tokens object*

Description

Get unique types of tokens from a tokens object.

Usage

```
types(x)
```

Arguments

x	a tokens object

See Also

featnames

Examples

```
toks <- tokens(data_corpus_inaugural)
head(types(toks), 20)
```

Index

∗ **bootstrap**
bootstrap_dfm, 11
∗ **character**
corpus_segment, 25
corpus_trim, 28
∗ **corpus**
corpus, 17
corpus_chunk, 20
corpus_group, 21
corpus_reshape, 22
corpus_sample, 23
corpus_segment, 25
corpus_subset, 27
corpus_trim, 28
docnames, 59
docvars, 60
meta, 69
∗ **data**
data_char_sampletext, 30
data_char_ukimmig2010, 30
data_corpus_inaugural, 31
data_dfm_lbgexample, 32
data_dictionary_LSD2015, 32
∗ **dfm**
as.matrix.dfm, 10
bootstrap_dfm, 11
dfm, 34
dfm_lookup, 38
dfm_match, 40
dfm_sample, 42
dfm_select, 43
dfm_subset, 46
dfm_tfidf, 48
dfm_weight, 52
docfreq, 57
docnames, 59
featfreq, 65
print-methods, 74
∗ **experimental**
bootstrap_dfm, 11
∗ **tokens**
tokens, 80
tokens_annotate, 84
tokens_chunk, 85
tokens_group, 88
tokens_lookup, 89
tokens_sample, 96
tokens_segment, 98
tokens_split, 101
tokens_subset, 102
tokens_xptr, 108
∗ **weighting**
dfm_tfidf, 48
docfreq, 57
featfreq, 65
[, *55*
[.corpus(), *19*
[[, *55*
$.corpus *(*docvars*)*, 60
$.dfm *(*docvars*)*, 60
$.tokens *(*docvars*)*, 60
$<-.corpus *(*docvars*)*, 60
$<-.dfm *(*docvars*)*, 60
$<-.tokens *(*docvars*)*, 60

as.character.corpus, 4
as.character.corpus(), *19*
as.character.tokens *(*as.list.tokens*)*, 8
as.corpus *(*as.character.corpus*)*, 4
as.data.frame.dfm(), *6*
as.data.frame.kwic *(*kwic*)*, 68
as.dfm, 5
as.dfm(), *35*
as.dictionary, 6
as.dictionary(), *55, 56*
as.fcm, 7
as.list(), *55, 56*
as.list.tokens, 8
as.list.tokens(), *83*

as.matrix.dfm, 10
as.matrix.dfm(), *6*
as.matrix.tokens *(*as.list.tokens*)*, 8
as.phrase *(*phrase*)*, 73
as.phrase(), *74*
as.tensor *(*as.list.tokens*)*, 8
as.tokens *(*as.list.tokens*)*, 8
as.tokens(), *82, 83*
as.tokens_xptr *(*tokens_xptr*)*, 108
as.yaml, 10

base::print(), *75*
base::tolower(), *13*
bootstrap_dfm, 11
bootstrap_dfm(), *40*

c.tokens, *8*
cbind.dfm(), *35*
char_keep *(*char_select*)*, 12
char_ngrams *(*tokens_ngrams*)*, 93
char_remove *(*char_select*)*, 12
char_segment *(*corpus_segment*)*, 25
char_select, 12
char_tolower, 13
char_toupper *(*char_tolower*)*, 13
char_trim *(*corpus_trim*)*, 28
char_wordstem *(*tokens_wordstem*)*, 106
char_wordstem(), *77*, *107*
character, *4*, *12, 13*, *17*, *81*, *107*
concat, 14
concat(), *14*, *83*
concatenator *(*concat*)*, 14
convert, 15
convert(), *6*
corpus, *4*, *15*, 17, *17*, *19*, *21, 22*, *24, 25*, *28, 29*, *31*, *59–61*, *68–71*, *78*, *81*
corpus_chunk, 20
corpus_group, 21
corpus_reshape, 22
corpus_reshape(), *26, 27*, *59*
corpus_sample, 23
corpus_segment, 25
corpus_segment(), *26*, *59*, *98*
corpus_subset, 27
corpus_trim, 28

data.frame, *5*, *17*
data_char_sampletext, 30
data_char_ukimmig2010, 30
data_corpus_inaugural, 31
data_dfm_lbgexample, 32
data_dictionary_LSD2015, 32
dfm, *5–7*, *10, 11*, *15*, *32*, 34, *34–38*, *40–47*, *51–54*, *57*, *59–61*, *63*, *65, 66*, *69–72*, *86*, *105*, *107, 108*
dfm(), *11*, *46*, *108*
dfm_compress, 35
dfm_group, 37
dfm_group(), *48*, *53*
dfm_keep *(*dfm_select*)*, 43
dfm_lookup, 38
dfm_lookup(), *9*
dfm_match, 40
dfm_match(), *45*
dfm_remove *(*dfm_select*)*, 43
dfm_replace, 41
dfm_sample, 42
dfm_sample(), *52*
dfm_select, 43
dfm_select(), *35*, *40*, *47*, *52*
dfm_smooth *(*dfm_weight*)*, 52
dfm_sort, 46
dfm_subset, 46
dfm_tfidf, 48
dfm_tfidf(), *65*
dfm_tolower, 49
dfm_tolower(), *35*
dfm_toupper *(*dfm_tolower*)*, 49
dfm_trim, 50
dfm_trim(), *45*, *106*
dfm_weight, 52
dfm_weight(), *37*, *48*, *65*
dfm_wordstem *(*tokens_wordstem*)*, 106
dfm_wordstem(), *77*, *107*
dictionary, *6*, *10*, *12*, *25*, *32, 33*, *38, 39*, *41*, *44*, 54, *67–69*, *73*, *84*, *87*, *90*, *95*, *98*, *100*
dictionary(), *6*
docfreq, *48*, 57
docfreq(), *48*, *54*
docid *(*docnames*)*, 59
docnames, *36*, 59
docnames(), *19*
docnames<- *(*docnames*)*, 59
DocumentTermMatrix, *5*, *15*
docvars, *17*, *22*, *28*, *36*, *38*, *46*, 60, *61*, *89*, *98*, *102*

docvars(), *19*
docvars<- *(*docvars*)*, 60

fcm, *7*, *35, 36*, *43–45*, *50*, *54*, 62, *63–65*
fcm(), *63*
fcm_compress *(*dfm_compress*)*, 35
fcm_keep *(*dfm_select*)*, 43
fcm_remove *(*dfm_select*)*, 43
fcm_select *(*dfm_select*)*, 43
fcm_sort, 64
fcm_sort(), *65*
fcm_tolower *(*dfm_tolower*)*, 49
fcm_toupper *(*dfm_tolower*)*, 49
featfreq, 65
featnames, *36*, *51*, 66, *105*, *110*
featnames(), *52*, *59*
file, *55*

getStemLanguages, *107*
groups, *59*

iconv, *55*
index, 66, *68*
index(), *67*
is.collocations, 67
is.corpus *(*as.character.corpus*)*, 4
is.dfm *(*as.dfm*)*, 5
is.dictionary *(*as.dictionary*)*, 6
is.dictionary(), *55, 56*
is.fcm *(*fcm*)*, 62
is.index *(*index*)*, 66
is.kwic *(*kwic*)*, 68
is.phrase *(*phrase*)*, 73
is.tokens *(*as.list.tokens*)*, 8
is.tokens_xptr *(*tokens_xptr*)*, 108

jsonlite::toJSON(), *16*

kwic, *17*, 68
kwic(), *17*, *100*

lda.collapsed.gibbs.sampler, *15*
list, *55*

Matrix, *5*, *7*
matrix, *5*, *7*
meta, *17*, 69
meta(), *18, 19*
meta<- *(*meta*)*, 69

ndoc, 70
ndoc(), *19*
nfeat *(*ndoc*)*, 70
nfeat(), *63*
nsentence, 71
ntoken, 72
ntoken(), *71*
ntype *(*ntoken*)*, 72
ntype(), *71*

options(), *76*

pattern, *12*, *25*, *41*, *44*, *66–68*, *87*, *95*, *98*, *100*
pattern matches, *33*
pattern(), *73*
phrase, 73
phrase(), *69*, *73*, *87*
print,dfm-method *(*print-methods*)*, 74
print,dictionary2-method *(*print-methods*)*, 74
print,fcm-method *(*print-methods*)*, 74
print-methods, *69*, 74
print.corpus *(*print-methods*)*, 74
print.dfm *(*print-methods*)*, 74
print.dictionary *(*print-methods*)*, 74
print.kwic *(*print-methods*)*, 74
print.tokens *(*print-methods*)*, 74

quanteda-package, *31*
quanteda_options, 76
quanteda_options(), *75, 76*
quantile(), *51*, *105*

rbind.dfm(), *35*

sample, *43*, *97*
segid *(*docnames*)*, 59
SimpleCorpus, *17*
spacy_parse, *78*
spacy_tokenize, *78*
spacyr-methods, 78
sparsity, 79
stri_opts_brkiter, *72*
stri_split_boundaries, *83*
stri_split_boundaries(x, type = word), *82*
stringi::stri_opts_brkiter(), *77*
subset.data.frame(), *28*, *46*, *102, 103*
substitute, *89*, *97*

TermDocumentMatrix, *5*, *15*
TermDocumentMatrix(), *15*
textmodels, 79
textplots, 79
textstats, 80
tokenize_word4(), *82*
tokens, *8, 9*, *14*, *20*, *34*, *54*, *57*, *59–61*, *63*, *65*, *67–72*, *75*, 80, *81*, *84–90*, *92*, *95*, *97–105*, *107, 108*, *110*
tokens(), *23*, *34*, *77*
tokens_annotate, 84
tokens_chunk, 85
tokens_chunk(), *21*
tokens_compound, *82*, 86
tokens_compound(), *14*, *83*, *101*
tokens_group, 88
tokens_keep *(*tokens_select*)*, 99
tokens_lookup, 89
tokens_lookup(), *9*, *14*, *39*, *83*, *95*
tokens_match, 92
tokens_ngrams, 93
tokens_ngrams(), *83*, *94*
tokens_remove *(*tokens_select*)*, 99
tokens_remove(), *34*
tokens_replace, 95
tokens_sample, 96
tokens_segment, 98
tokens_segment(), *86*
tokens_select, 99
tokens_skipgrams *(*tokens_ngrams*)*, 93
tokens_skipgrams(), *83*, *94*
tokens_split, 101
tokens_subset, 102
tokens_tolower, 104
tokens_toupper *(*tokens_tolower*)*, 104
tokens_trim, 104
tokens_wordstem, 106
tokens_wordstem(), *77*, *107*
tokens_xptr, *57*, *59*, *65*, *70, 71*, 108
topfeatures, 108
types, *51*, *105*, 110

valuetype, *12*, *25*, *39*, *44*, *55*, *67, 68*, *84*, *87*, *90*, *95*, *98*, *100*, *102*
VCorpus, *17*

wordStem, *106, 107*