

# Package ‘quanteda’

November 27, 2015

**Type** Package

**Title** Quantitative Analysis of Textual Data

**Author** Kenneth Benoit [aut, cre],  
Paul Nulty [aut],  
Pablo Barberá [ctb],  
Kohei Watanabe [ctb],  
Benjamin Lauderdale [ctb]

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

**Version** 0.9.0-1

**Date** 2015-11-26

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

**License** GPL-3

**Depends** R (>= 3.2.0)

**Imports** methods, utils, stats, Matrix (>= 1.1), data.table (>= 1.9.3),  
SnowballC, wordcloud, proxy, parallel, Rcpp, ca, stringi

**LinkingTo** Rcpp, RcppArmadillo

**Suggests** knitr, lda, topicmodels, jsonlite (>= 0.9.10), streamR, tm  
(>= 0.6), slam, ggplot2, XML, testthat, RColorBrewer

**URL** <http://github.com/kbenoit/quanteda>

**Encoding** UTF-8

**BugReports** <https://github.com/kbenoit/quanteda/issues>

**LazyData** TRUE

**VignetteBuilder** knitr

**Collate** 'RcppExports.R' 'dictionaries.R' 'collocations.R'  
'converters.R' 'corpus.R' 'dataDocs.R' 'describe-texts.R'  
'dfm-classes.R' 'dfm-main.R' 'dfm-methods.R' 'distance.R'  
'encoding.R' 'kwic.R' 'lexdiv.R' 'ngrams.R' 'plots.R'  
'quanteda-package.R' 'readability.R' 'resample.R' 'settings.R'  
'stopwords.R' 'syllables.R' 'textfile.R' 'textmodel-ca.R'

'textmodel-generics.R' 'textmodel-wordfish.R'  
 'textmodel-wordscores.R' 'toLower.R' 'tokenize.R'  
 'tokenize\_outtakes.R' 'wordstem.R'

**RoxygenNote** 5.0.1

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2015-11-27 11:41:19

## R topics documented:

quanteda-package . . . . .	3
applyDictionary . . . . .	4
cbind.dfm . . . . .	5
changeunits . . . . .	6
collocations . . . . .	7
convert . . . . .	9
corpus . . . . .	11
corpusSource-class . . . . .	14
dfm . . . . .	15
dfm-class . . . . .	18
dictionary . . . . .	22
docfreq . . . . .	23
docnames . . . . .	24
docvars . . . . .	25
encodedTexts . . . . .	26
encoding . . . . .	26
exampleString . . . . .	27
features . . . . .	28
head.dfm . . . . .	28
ie2010Corpus . . . . .	29
inaugCorpus . . . . .	30
kwic . . . . .	30
LBGexample . . . . .	32
lexdiv . . . . .	32
metacorporus . . . . .	35
metadoc . . . . .	36
ndoc . . . . .	37
ngrams . . . . .	38
nsentence . . . . .	40
ntoken . . . . .	41
phrasetotoken . . . . .	42
plot.dfm . . . . .	43
plot.kwic . . . . .	44
print.dfm . . . . .	45
print.tokenizedTexts . . . . .	46
readability . . . . .	46

removeFeatures . . . . .	48
sample.corpus . . . . .	49
scrabble . . . . .	51
segment . . . . .	52
selectFeatures . . . . .	53
settings . . . . .	55
show,dictionary-method . . . . .	56
similarity . . . . .	56
sort.dfm . . . . .	58
stopwords . . . . .	59
subset.corpus . . . . .	60
summary.corpus . . . . .	60
syllables . . . . .	61
textfile . . . . .	63
textmodel . . . . .	65
textmodel_ca . . . . .	67
textmodel_fitted-class . . . . .	68
textmodel_wordfish . . . . .	68
textmodel_wordscores . . . . .	70
texts . . . . .	72
tokenize . . . . .	73
toLower . . . . .	76
topfeatures . . . . .	77
trim . . . . .	78
ukimmigTexts . . . . .	79
weight . . . . .	80
wordlists . . . . .	82
wordstem . . . . .	82
<b>Index</b>	<b>84</b>

---

quanteda-package	<i>An R package for the quantitative analysis of textual data.</i>
------------------	--

---

## Description

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

More detailed description, and some examples, to go here.

## Author(s)

Ken Benoit and Paul Nulty

---

applyDictionary	<i>apply a dictionary or thesarus to an object</i>
-----------------	--

---

### Description

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

### Usage

```
applyDictionary(x, dictionary, ...)

## S3 method for class 'dfm'
applyDictionary(x, dictionary, exclusive = TRUE,
  valuetype = c("glob", "regex", "fixed"), case_insensitive = TRUE,
  capkeys = !exclusive, verbose = TRUE, ...)
```

### Arguments

x	object to which dictionary or thesaurus will be supplied
dictionary	the <a href="#">dictionary</a> -class object that will be applied to x
...	not used
exclusive	if TRUE, remove all features not in dictionary, otherwise, replace values in dictionary with keys while leaving other features unaffected
valuetype	how to interpret dictionary values: "glob" for "glob"-style wildcard expressions (the format used in Wordstat and LIWC formatted dictionary values); "regex" for regular expressions; or "fixed" for exact matching (entire words, for instance)
case_insensitive	ignore the case of dictionary values if TRUE
capkeys	if TRUE, convert dictionary or thesaurus keys to uppercase to distinguish them from other features
verbose	print status messages if TRUE

### Value

an object of the type passed with the value-matching features replaced by dictionary keys

### Note

Selecting only features defined in a "dictionary" is traditionally known in text analysis as a *dictionary method*, even though technically this they operate more like a thesarus. If a more truly thesaurus-like application is desired, set `keeponly = FALSE` to convert features defined as values in a dictionary into their keys, while keeping all other features.

## Examples

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

myDfm

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

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

# fixed format: no pattern matching
applyDictionary(myDfm, myDict, valuetype = "fixed")
applyDictionary(myDfm, myDict, valuetype = "fixed", case_insensitive = FALSE)
```

---

cbind.dfm	<i>combine dfm object by columns</i>
-----------	--------------------------------------

---

## Description

Combine two or more dfm objects by columns. An implementation of [cbind](#) for [dfm](#) objects. Calls [cbind2](#) defined for object classes in the **Matrix** package.

## Usage

```
cbind.dfm(x, y, ...)
```

## Arguments

x	a <a href="#">dfm</a> object
y	a second <a href="#">dfm</a> object to be joined column-wise to the first
...	optional arguments for methods

## Value

A dfm object with combined features from input dfm objects. The attributes of this new dfm are not currently preserved.

## Examples

```
dfm1 <- dfm("This is a sample text.", verbose = FALSE)
dfm2 <- dfm("one two three", verbose = FALSE)
cbind(dfm1, dfm2)
```

---

changeunits

*change the document units of a corpus*

---

## Description

For a corpus, recast the documents down or up a level of aggregation. "Down" would mean going from documents to sentences, for instance. "Up" means from sentences back to documents. This makes it easy to reshape a corpus from a collection of documents into a collection of sentences, for instance.

## Usage

```
changeunits(x, ...)

## S3 method for class 'corpus'
changeunits(x, to = c("sentences", "paragraphs",
  "documents"), ...)
```

## Arguments

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

## Value

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

## Examples

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

# example with inaugural corpus speeches
(mycorpus2 <- subset(inaugCorpus, Year>2004))
paragCorpus <- changeunits(mycorpus2, to="paragraphs")
paragCorpus
```

```
summary(paragCorpus, 100, showmeta=TRUE)
## Note that Bush 2005 is recorded as a single paragraph because that text used a single
## \n to mark the end of a paragraph.
```

collocations

*Detect collocations from text*

## Description

Detects collocations (currently, bigrams and trigrams) from texts or a corpus, returning a data.frame of collocations and their scores, sorted in descending order of the association measure. Words separated by punctuation delimiters are not counted as adjacent and hence are not eligible to be collocations.

## Usage

```
collocations(x, ...)

## S3 method for class 'corpus'
collocations(x, method = c("lr", "chi2", "pmi", "dice",
  "all"), size = 2, n = NULL, spanPunct = FALSE, ...)

## S3 method for class 'character'
collocations(x, method = c("lr", "chi2", "pmi", "dice",
  "all"), size = 2, n = NULL, spanPunct = FALSE, ...)

## S3 method for class 'tokenizedTexts'
collocations(x, method = c("lr", "chi2", "pmi",
  "dice", "all"), size = 2, n = NULL, spanPunct = FALSE, ...)
```

## Arguments

x	a text, a character vector of texts, or a corpus
...	additional parameters passed to <a href="#">tokenize</a> . If wanted to include collocations separated by punctuation, then you can use this to send <code>removePunct = TRUE</code> to <a href="#">tokenize</a> .
method	association measure for detecting collocations. Let $i$ index documents, and $j$ index features, $n_{ij}$ refers to observed counts, and $m_{ij}$ the expected counts in a collocations frequency table of dimensions $(J - size + 1)^2$ . Available measures are computed as:  "lr" The likelihood ratio statistic $G^2$ , computed as:

$$2 * \sum_i \sum_j (n_{ij} * \log \frac{n_{ij}}{m_{ij}})$$

"chi2" Pearson's  $\chi^2$  statistic, computed as:

$$\sum_i \sum_j \frac{(n_{ij} - m_{ij})^2}{m_{ij}}$$

"pmi" point-wise mutual information score, computed as  $\log n_{11}/m_{11}$

"dice" the Dice coefficient, computed as  $n_{11}/n_{1.} + n_{.1}$

"all" returns all of the above

size	length of the collocation. Only bigram (n=2) and trigram (n=3) collocations are implemented so far. Can be c(2,3) (or 2:3) to return both bi- and tri-gram collocations.
n	the number of collocations to return, sorted in descending order of the requested statistic, or $G^2$ if none is specified.
spanPunct	if FALSE, then collocations will not span punctuation marks, so that for instance <i>marks, so</i> is not a collocation of <i>marks so</i> . If TRUE, do not handle punctuation specially.

### Value

A data.table of collocations, their frequencies, and the computed association measure(s).

### Author(s)

Kenneth Benoit

### References

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

### See Also

[ngrams](#)

### Examples

```
txt <- c("This is software testing: looking for (word) pairs!
        This [is] a software testing again. For.",
        "Here: this is more Software Testing, looking again for word pairs.")
collocations(txt)
collocations(txt, removePunct = TRUE)
collocations(txt, size=2:3)
removeFeatures(collocations(txt, size=2:3), stopwords("english"))

collocations("@textasdata We really, really love the #quanteda package - thanks!!")
collocations("@textasdata We really, really love the #quanteda package - thanks!!",
              removeTwitter = TRUE)

collocations(inaugTexts[49:57], n=10)
collocations(inaugTexts[49:57], method="all", n=10)
```



```
collocations(inaugTexts[49:57], method="chi2", size=3, n=10)
collocations(subset(inaugCorpus, Year>1980), method="pmi", size=3, n=10)
```

---

convert

*convert a dfm to a non-quanteda format*


---

## Description

Convert a quanteda [dfm-class](#) object to a format useable by other text analysis packages. The general function `convert` provides easy conversion from a dfm to the document-term representations used in all other text analysis packages for which conversions are defined. To make the usage as consistent as possible with other packages, however, quanteda also provides direct conversion functions in the idiom of the foreign packages, for example `as.wfm` to coerce a dfm into the wfm format from the **austin** package, and `quantedaformat2dtm` for using a dfm with the **topicmodels** package.

## Usage

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

## S3 method for class 'dfm'
convert(x, to = c("lda", "tm", "stm", "austin", "topicmodels"),
  ...)

as.wfm(x)

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

as.DocumentTermMatrix(x, ...)

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

dfm2ldaformat(x)

## S3 method for class 'dfm'
dfm2ldaformat(x)

quantedaformat2dtm(x)

## S3 method for class 'dfm'
quantedaformat2dtm(x)
```

## Arguments

x                      dfm to be converted

to	target conversion format, consisting of the name of the package into whose document-term matrix representation the dfm will be converted:
"	lda" a list with components "documents" and "vocab" as needed by <a href="#">lda.collapsed.gibbs.sampler</a> from the <b>lda</b> package
"	tm" a <a href="#">DocumentTermMatrix</a> from the <b>tm</b> package
"	stm" the format for the <b>stm</b> package
"	austin" the wfm format from the <b>austin</b> package
"	topicmodels" the "dtm" format as used by the <b>topicmodels</b> package
...	not used here

## Details

We recommend using `convert()` rather than the specific functions. In fact, it's worth considering whether we should simply remove all of them and **only** support calling these through `convert()`.

We may also use this function, eventually, for converting other classes of objects such as a `'corpus'` or `'tokenizedList'`.

`as.wfm` converts a quanteda [dfm](#) into the wfm format used by the **austin** package.

`as.DocumentTermMatrix` will convert a quanteda [dfm](#) into the **tm** package's [DocumentTermMatrix](#) format.

`dfm2ldaformat` provides converts a [dfm](#) into the list representation of terms in documents used by the **lda** package.

`quantedaformat2dtm` provides converts a [dfm](#) into the sparse simple triplet matrix representation of terms in documents used by the **topicmodels** package.

## 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.

`dfm2ldaformat` returns a list with components "documents" and "vocab" as needed by [lda.collapsed.gibbs.sampler](#).

`quantedaformat2dtm` returns a "dtm" sparse matrix object for use with the **topicmodels** 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.

## Examples

```
mycorpus <- subset(inaugCorpus, Year>1970)
quantdfm <- dfm(mycorpus, verbose=FALSE)

# austin's wfm format
austindfm <- as.wfm(quantdfm)
identical(austindfm, convert(quantdfm, to="austin"))
```

```

# tm's DocumentTermMatrix format
tmdfm <- as.DocumentTermMatrix(quantdfm)
str(tmdfm)

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

# topicmodels package format
topicmodelsdfm <- quantdaformat2dtm(quantdfm)
identical(topicmodelsdfm, convert(quantdfm, to="topicmodels"))

# lda package format
ldadfm <- convert(quantdfm, to="lda")
str(ldadfm)
identical(ldadfm, stmdfm[1:2])
# calling dfm2ldaformat directly
ldadfm <- dfm2ldaformat(quantdfm)
str(ldadfm)

```

---

corpus

*constructor for corpus objects*


---

## Description

Creates a corpus from a document source. The current available document sources are:

- a character vector (as in R class `char`) of texts;
- a [corpusSource-class](#) object, constructed using [textfile](#);
- a **tm** [VCorpus](#) class corpus object, meaning that anything you can use to create a **tm** corpus, including all of the tm plugins plus the built-in functions of tm for importing pdf, Word, and XML documents, can be used to create a quantda [corpus](#).

Corpus-level meta-data can be specified at creation, containing (for example) citation information and notes, as can document-level variables and document-level meta-data.

## Usage

```

corpus(x, ...)

## S3 method for class 'character'
corpus(x, enc = NULL, encTo = "UTF-8",
       docnames = NULL, docvars = NULL, source = NULL, notes = NULL,
       citation = NULL, ...)

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

```

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

is.corpus(x)

## S3 method for class 'corpus'
c1 + c2

## S3 method for class 'corpus'
x[i, j = NULL, ..., drop = TRUE]
```

## Arguments

x	a source of texts to form the documents in the corpus, a character vector or a <a href="#">corpusSource-class</a> object created using <a href="#">textfile</a> .
...	additional arguments
enc	<p>a string specifying the input encoding for texts in the corpus. Must be a valid entry in <a href="#">stri_enc_list()</a>, since the code in <code>corpus.character</code> will convert this to <code>encTo</code> using <a href="#">stri_encode</a>. We recommend that you do <b>not</b> use <code>enc</code>, since if left NULL (the default) then <code>corpus()</code> will detect the input encoding(s) and convert automatically.</p> <p>Currently only one input encoding can be specified for a collection of input texts, meaning that you should not mix input text encoding types in a single corpus call. However if you suspect multiple encodings, omit the <code>enc</code> argument and <code>corpus()</code> will detect and convert each file automatically.</p>
encTo	target encoding, default is UTF-8. Unless you have strong reasons to use an alternative encoding, we strongly recommend you leave this at its default. Must be a valid entry in <a href="#">stri_enc_list()</a>
docnames	Names to be assigned to the texts, defaults to the names of the character vector (if any), otherwise assigns "text1", "text2", etc.
docvars	A data frame of attributes that is associated with each text.
source	A string specifying the source of the texts, used for referencing.
notes	A string containing notes about who created the text, warnings, To Dos, etc.
citation	Information on how to cite the corpus.
c1	corpus one to be added
c2	corpus two to be added
i	index for documents or rows of document variables
j	index for column of document variables
drop	<p>if TRUE the result is coerced to the lowest possible dimension (see the examples). This only works for extracting elements, not for the replacement. See <a href="#">drop</a> for further details.</p>

## Details

The texts and document variables of corpus objects can also be accessed using index notation. Indexing a corpus object as a vector will return its text, equivalent to `texts(x)`. Indexing a corpus using two indexes (integers or column names) will return the document variables, equivalent to `docvars(x)`.

The `+` operator for a corpus object will combine two corpus objects, resolving any non-matching `docvars` or `metadoc` fields by making them into NA values for the corpus lacking that field. Corpus-level meta data is concatenated, except for source and notes, which are stamped with information pertaining to the creation of the new joined corpus.

There are some issues that need to be addressed in future revisions of `quanteda` concerning the use of factors to store document variables and meta-data. Currently most or all of these are not recorded as factors, because we use `stringsAsFactors=FALSE` in the `data.frame` calls that are used to create and store the document-level information, because the texts should always be stored as character vectors and never as factors.

## Value

A corpus class object containing the original texts, document-level variables, document-level meta-data, corpus-level metadata, and default settings for subsequent processing of the corpus. A corpus consists of a list of elements described below, although these should only be accessed through accessor and replacement functions, not directly (since the internals may be subject to change). The structure of a corpus classed list object is:

<code>\$documents</code>	A data frame containing the document level information, consisting of <code>texts</code> , user-named <code>docvars</code> variables describing attributes of the documents, and <code>metadoc</code> document-level metadata whose names begin with an underscore character, such as <code>_language</code> .
<code>\$metadata</code>	A named list set of corpus-level meta-data, including source and created (both generated automatically unless assigned), notes, and citation.
<code>\$settings</code>	Settings for the corpus which record options that govern the subsequent processing of the corpus when it is converted into a document-feature matrix ( <code>dfm</code> ). See <a href="#">settings</a> .
<code>\$tokens</code>	An indexed list of tokens and types tabulated by document, including information on positions. Not yet fully implemented.

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

## Note

When `x` is a `VCorpus` object, the fixed metadata fields from that object are imported as document-level metadata. Currently no corpus-level metadata is imported, but we will add that soon.

## Author(s)

Kenneth Benoit and Paul Nulty

**See Also**

[docvars](#), [metadoc](#), [metacorporus](#), [settings](#), [texts](#)

**Examples**

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

# create a corpus from texts and assign meta-data and document variables
ukimmigCorpus <- corpus(ukimmigTexts,
                        docvars = data.frame(party=names(ukimmigTexts)),
                        encTo = "UTF-16")

corpus(texts(ie2010Corpus))
## Not run: # the fifth column of this csv file is the text field
mytexts <- textfile("http://www.kenbenoit.net/files/text_example.csv", textField = 5)
mycorp <- corpus(mytexts)
mycorp2 <- corpus(textfile("http://www.kenbenoit.net/files/text_example.csv", textField = "Title"))
identical(texts(mycorp), texts(mycorp2))
identical(docvars(mycorp), docvars(mycorp2))

## End(Not run)
# import a tm VCorpus
if ("tm" %in% rownames(installed.packages())) {
  data(crude, package = "tm") # load in a tm example VCorpus
  mytmCorpus <- corpus(crude)
  summary(mytmCorpus, showmeta=TRUE)

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

  tmCorp <- tm::VCorpus(tm::VectorSource(inaugTexts[49:57]))
  quantCorp <- corpus(tmCorp)
  summary(quantCorp)
}
```

---

corpusSource-class      *corpus source classes*

---

**Description**

The corpusSource virtual class is a parent class for more specific corpus source objects.

**Usage**

```
## S4 method for signature 'corpusSource'
show(object)
```

**Arguments**

object corpusSource object to be printed

**Slots**

texts the texts that form the core of the corpus

docvars document variables in a data.frame

source source recorded for the corpus, based on type of source

created a time stamp

cachedfile if read to a temporary file, a string containing the location of the temporary file

---

dfm	<i>create a document-feature matrix</i>
-----	---

---

**Description**

Create a sparse matrix document-feature matrix from a corpus or a vector of texts. The sparse matrix construction uses the **Matrix** package, and is both much faster and much more memory efficient than the corresponding dense (regular matrix) representation. For details on the structure of the dfm class, see [dfm-class](#).

**Usage**

```
dfm(x, ...)
```

```
## S3 method for class 'character'
```

```
dfm(x, verbose = TRUE, toLower = TRUE,
    removeNumbers = TRUE, removePunct = TRUE, removeSeparators = TRUE,
    removeTwitter = FALSE, stem = FALSE, ignoredFeatures = NULL,
    keptFeatures = NULL, matrixType = c("sparse", "dense"),
    language = "english", thesaurus = NULL, dictionary = NULL,
    valuetype = c("glob", "regex", "fixed"), dictionary_regex = FALSE, ...)
```

```
## S3 method for class 'tokenizedTexts'
```

```
dfm(x, verbose = TRUE, toLower = TRUE,
    stem = FALSE, ignoredFeatures = NULL, keptFeatures = NULL,
    matrixType = c("sparse", "dense"), language = "english",
    thesaurus = NULL, dictionary = NULL, valuetype = c("glob", "regex",
    "fixed"), dictionary_regex = FALSE, ...)
```

```
## S3 method for class 'corpus'
```

```
dfm(x, verbose = TRUE, groups = NULL, ...)
```

```
is.dfm(x)
```

```
as.dfm(x)
```

**Arguments**

<code>x</code>	corpus or character vector from which to generate the document-feature matrix
<code>...</code>	additional arguments passed to <a href="#">tokenize</a> , which can include for instance ngrams and concatenator for tokenizing multi-token sequences
<code>verbose</code>	display messages if TRUE
<code>toLower</code>	convert texts to lowercase
<code>removeNumbers</code>	remove numbers, see <a href="#">tokenize</a>
<code>removePunct</code>	remove numbers, see <a href="#">tokenize</a>
<code>removeSeparators</code>	remove separators (whitespace), see <a href="#">tokenize</a>
<code>removeTwitter</code>	if FALSE, preserve # and @ characters, see <a href="#">tokenize</a>
<code>stem</code>	if TRUE, stem words
<code>ignoredFeatures</code>	a character vector of user-supplied features to ignore, such as "stop words". To access one possible list (from any list you wish), use <a href="#">stopwords()</a> . The pattern matching type will be set by <code>valuetype</code> . For behaviour of <code>ingoredFeatures</code> with <code>ngrams &gt; 1</code> , see Details.
<code>keptFeatures</code>	a use supplied regular expression defining which features to keep, while excluding all others. This can be used in lieu of a dictionary if there are only specific features that a user wishes to keep. To extract only Twitter usernames, for example, set <code>keptFeatures = "@*"</code> and make sure that <code>removeTwitter = FALSE</code> as an additional argument passed to <a href="#">tokenize</a> . Note: <code>keptFeatures = "^@\\w+\\b"</code> would be the regular expression version of this matching pattern. The pattern matching type will be set by <code>valuetype</code> .
<code>matrixType</code>	deprecated, used to produce a dense matrix if dense, but this was removed in 0.8.2. All dfm objects are now created as a sparse matrix of class <code>dgCMatrix</code> from the <a href="#">Matrix</a> package.
<code>language</code>	Language for stemming. Choices are danish, dutch, english, finnish, french, german, hungarian, italian, norwegian, porter, portuguese, romanian, russian, spanish, swedish, turkish.
<code>thesaurus</code>	A list of character vector "thesaurus" entries, in a dictionary list format, which can also include regular expressions if <code>dictionary_regex</code> is TRUE (see examples). Note that unlike dictionaries, each entry in a thesaurus key must be unique, otherwise only the first match in the list will be used. Thesaurus keys are converted to upper case to create a feature label in the dfm, as a reminder that this was not a type found in the text, but rather the label of a thesaurus key.
<code>dictionary</code>	A list of character vector dictionary entries, including regular expressions (see examples)
<code>valuetype</code>	fixed for words as is; "regex" for regular expressions; or "glob" for "glob"-style wildcard. Glob format is the default. See <a href="#">selectFeatures</a> .
<code>dictionary_regex</code>	TRUE means the dictionary is already in regular expression format, otherwise it will be converted from the "glob" format. This is a legacy argument that will soon be phased out in favour of <code>valuetype</code> .



**groups** character vector containing the names of document variables for aggregating documents

## Details

The default behavior for `ignoredFeatures` when constructing ngrams using `dfm(x, ngrams > 1)` is to remove any ngram that contains any item in `ignoredFeatures`. If you wish to remove these before constructing ngrams, you will need to first tokenize the texts with `ngrams`, then remove the features to be ignored, and then construct the dfm using this modified tokenization object. See the code examples for an illustration.

`is.dfm` returns TRUE if and only if its argument is a [dfm](#).

`as.dfm` coerces a matrix or data.frame to a dfm

## Value

A [dfm-class](#) object containing a sparse matrix representation of the counts of features by document, along with associated settings and metadata.

## Author(s)

Kenneth Benoit

## Examples

```
# why we phased out dense matrix dfm objects
(size1 <- object.size(dfm(inaugTexts, verbose = FALSE)))
(size2 <- object.size(as.matrix(dfm(inaugTexts, verbose = FALSE))))
cat("Compacted by ", round(as.numeric((1-size1/size2)*100), 1), "%.\n", sep="")

# for a corpus
mydfm <- dfm(subset(inaugCorpus, Year>1980))
mydfm <- dfm(subset(inaugCorpus, Year>1980), toLower=FALSE)

# grouping documents by docvars in a corpus
mydfmGrouped <- dfm(subset(inaugCorpus, Year>1980), groups = "President")

# with English stopwords and stemming
dfmsInaug2 <- dfm(subset(inaugCorpus, Year>1980),
  ignoredFeatures=stopwords("english"), stem=TRUE)
# works for both words in ngrams too
dfm("Banking industry", stem = TRUE, ngrams = 2, verbose = FALSE)

# with dictionaries
mycorpus <- subset(inaugCorpus, Year>1900)
mydict <- list(christmas=c("Christmas", "Santa", "holiday"),
  opposition=c("Opposition", "reject", "notincorporus"),
  taxing="taxing",
  taxation="taxation",
  taxregex="tax*",
  country="united states")
dictDfm <- dfm(mycorpus, dictionary=mydict)
```

```
dictDfm

# with the thesaurus feature
mytexts <- c("The new law included a capital gains tax, and an inheritance tax.",
            "New York City has raised a taxes: an income tax and a sales tax.")
mydict <- dictionary(list(tax=c("tax", "income tax", "capital gains tax", "inheritance tax")))
dfm(phrasetotoken(mytexts, mydict), thesaurus = lapply(mydict, function(x) gsub("\\s", "_", x)))
# pick up "taxes" with "tax" as a regex
dfm(phrasetotoken(mytexts, mydict), thesaurus = list(anytax = "tax"), valuetype = "regex")

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

## removing stopwords before constructing ngrams
tokensAll <- tokenize(toLower(testText), removePunct = TRUE)
tokensNoStopwords <- removeFeatures(tokensAll, stopwords("english"))
tokensNgramsNoStopwords <- ngrams(tokensNoStopwords, 2)
features(dfm(tokensNgramsNoStopwords, ngrams = 1:2))

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

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

---

dfm-class

*Virtual class "dfm" for a document-feature matrix*


---

## Description

The dfm class of object is a type of [Matrix-class](#) object with additional slots, described below. **quanteda** uses two subclasses of the dfm class, depending on whether the object can be represented by a sparse matrix, in which case it is a dfmSparse class object, or if dense, then a dfmDense object. See Details.

## Usage

```
## S4 method for signature 'dfm'
```

```
t(x)

## S4 method for signature 'dfmSparse'
colSums(x, na.rm = FALSE, dims = 1L, ...)

## S4 method for signature 'dfmDense'
colSums(x, na.rm = FALSE, dims = 1L, ...)

## S4 method for signature 'dfmSparse'
rowSums(x, na.rm = FALSE, dims = 1L, ...)

## S4 method for signature 'dfmDense'
rowSums(x, na.rm = FALSE, dims = 1L, ...)

## S3 method for class 'dfm'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'dfmDense,index,index,missing'
x[i = NULL, j = NULL, ...,
  drop = FALSE]

## S4 method for signature 'dfmDense,index,index,logical'
x[i = NULL, j = NULL, ...,
  drop = FALSE]

## S4 method for signature 'dfmDense,index,missing,missing'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'dfmDense,index,missing,logical'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'dfmDense,missing,index,missing'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'dfmDense,missing,index,logical'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'dfmDense,missing,missing,missing'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'dfmDense,missing,missing,logical'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'dfmSparse,index,index,missing'
x[i = NULL, j = NULL, ...,
  drop = FALSE]

## S4 method for signature 'dfmSparse,index,index,logical'
```

```

x[i = NULL, j = NULL, ...,
  drop = FALSE]

## S4 method for signature 'dfmSparse,index,missing,missing'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'dfmSparse,index,missing,logical'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'dfmSparse,missing,index,missing'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'dfmSparse,missing,index,logical'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'dfmSparse,missing,missing,missing'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'dfmSparse,missing,missing,logical'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'dfmSparse,numeric'
e1 + e2

## S4 method for signature 'numeric,dfmSparse'
e1 + e2

## S4 method for signature 'dfmDense,numeric'
e1 + e2

## S4 method for signature 'numeric,dfmDense'
e1 + e2

## S4 method for signature 'dfm'
as.matrix(x)

## S4 method for signature 'dfm'
as.data.frame(x)

```

### Arguments

<code>x</code>	the dfm object
<code>na.rm</code>	if TRUE, omit missing values (including NaN) from the calculations
<code>dims</code>	ignored
<code>...</code>	additional arguments not used here
<code>i</code>	index for documents
<code>j</code>	index for features

drop	always set to FALSE
e1	first quantity in "+" operation for dfm
e2	second quantity in "+" operation for dfm

## Details

The dfm class is a virtual class that will contain one of two subclasses for containing the cell counts of document-feature matrixes: dfmSparse or dfmDense.

The dfmSparse class is a sparse matrix version of dfm-class, inheriting [dgCMatrix-class](#) from the **Matrix** package. It is the default object type created when feature counts are the object of interest, as typical text-based feature counts tend contain many zeroes. As long as subsequent transformations of the dfm preserve cells with zero counts, the dfm should remain sparse.

When the **Matrix** package implements sparse integer matrixes, we will switch the default object class to this object type, as integers are 4 bytes each (compared to the current numeric double type requiring 8 bytes per cell.)

The dfmDense class is a sparse matrix version of dfm-class, inheriting [dgeMatrix-class](#) from the **Matrix** package. dfm objects that are converted through weighting or other transformations into cells without zeroes will be automatically converted to the dfmDense class. This will necessarily be a much larger sized object than one of dfmSparse class, because each cell is recorded as a numeric (double) type requiring 8 bytes of storage.

## Slots

**settings** settings that govern corpus handling and subsequent downstream operations, including the settings used to clean and tokenize the texts, and to create the dfm. See [settings](#).

**weighting** the feature weighting applied to the dfm. Default is "frequency", indicating that the values in the cells of the dfm are simple feature counts. To change this, use the [weight](#) method.

**smooth** a smoothing parameter, defaults to zero. Can be changed using either the [smooth](#) or the [weight](#) methods.

**Dimnames** These are inherited from [Matrix-class](#) but are named docs and features respectively.

## See Also

[dfm](#)

## Examples

```
## Not run:
dfmSparse <- dfm(inaugTexts, verbose=FALSE)
str(as.matrix(dfmSparse))
class(as.matrix(dfmSparse))
dfmDense <- dfm(inaugTexts, verbose=FALSE, matrixType="dense")
str(as.matrix(dfmDense))
class(as.matrix(dfmDense))
identical(as.matrix(dfmSparse), as.matrix(dfmDense))

## End(Not run)
## Not run:
```

```
dfmSparse <- dfm(inaugTexts, verbose=FALSE)
str(as.data.frame(dfmSparse))
class(as.data.frame(dfmSparse))
dfmDense <- dfm(inaugTexts, verbose=FALSE, matrixType="dense")
str(as.data.frame(dfmDense))
class(as.data.frame(dfmDense))
identical(as.data.frame(dfmSparse), as.data.frame(dfmDense))

## End(Not run)
```

---

dictionary

*create a dictionary*


---

## Description

Create a quanteda dictionary, either from a list or by importing from a foreign format. Currently supported formats are the Wordstat and LIWC formats.

## Usage

```
dictionary(x = NULL, file = NULL, format = NULL, enc = "",
  tolower = TRUE, maxcats = 25)
```

## Arguments

x	a list of character vector dictionary entries, including regular expressions (see examples)
file	file identifier for a foreign dictionary
format	character identifier for the format of the foreign dictionary. Available options are: "wordstat" format used by Provalis Research's Wordstat software "LIWC" format used by the Linguistic Inquiry and Word Count software
enc	optional encoding value for reading in imported dictionaries. This uses the <a href="#">iconv</a> labels for encoding. See the "Encoding" section of the help for <a href="#">file</a> .
tolower	if TRUE, convert all dictionary functions to lower
maxcats	optional maximum categories to which a word could belong in a LIWC dictionary file, defaults to 10 (which is more than the actual LIWC 2007 dictionary uses). The default value of 10 is likely to be more than enough.

## Value

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

## Note

We will eventually change this to an S4 class with validators and additional methods.

## References

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

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

## See Also

[dfm](#)

## Examples

```
mycorpus <- subset(inaugCorpus, Year>1900)
mydict <-
  dictionary(list(christmas=c("Christmas", "Santa", "holiday"),
                 opposition=c("Opposition", "reject", "notin corpus"),
                 taxing="taxing",
                 taxation="taxation",
                 taxregex="tax*",
                 country="united states"))
dfm(mycorpus, dictionary=mydict)
## Not run:
# import the Laver-Garry dictionary from http://bit.ly/1FH2nvf
lgdict <- dictionary(file="http://www.kenbenoit.net/courses/essex2014qta/LaverGarry.cat",
                    format="wordstat")
dfm(inaugTexts, dictionary=lgdict)

# import a LIWC formatted dictionary
liwcdict <- dictionary(file = "http://www.kenbenoit.net/files/LIWC2001_English.dic",
                     format = "LIWC")
dfm(inaugTexts, dictionary=liwcdict)

## End(Not run)
```

---

docfreq

*get the document frequency of a feature*

---

## Description

For a [dfm-class](#) object, returns the number of documents in which a feature occurs more than a given frequency threshold. The default is greater than zero, meaning that a feature occurs at least once in a document.

## Usage

```
docfreq(object, threshold = 0)

## S4 method for signature 'dfmDense,numeric'
```

```

docfreq(object, threshold = 0)

## S4 method for signature 'dfmDense,missing'
docfreq(object, threshold = 0)

## S4 method for signature 'dfmSparse,numeric'
docfreq(object, threshold = 0)

## S4 method for signature 'dfmSparse,missing'
docfreq(object, threshold = 0)

## S4 method for signature 'dfm,numeric'
docfreq(object, threshold = 0)

## S4 method for signature 'dfm,missing'
docfreq(object, threshold = 0)

```

### Arguments

object	a <a href="#">dfm-class</a> document-feature matrix
threshold	numeric value of the threshold <i>above which</i> 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.

### Value

a numeric vector of document frequencies for each feature

### Examples

```

mydfm <- dfm(inaugTexts[1:2], verbose = FALSE)
docfreq(mydfm[, 1:20])

```

---

docnames	<i>get or set document names</i>
----------	----------------------------------

---

### Description

Extract the document names from a corpus or a document-feature matrix. of the [dfm](#) object.

docnames queries the document names of a corpus or a dfm

docnames <- assigns new values to the document names of a corpus. (Does not work for dfm objects, whose document names are fixed.)



**Usage**

```
docnames(x)

## S3 method for class 'corpus'
docnames(x)

docnames(x) <- value

## S3 method for class 'dfm'
docnames(x)
```

**Arguments**

x	the object with docnames
value	a character vector of the same length as x

**Value**

docnames returns a character vector of the document names  
docnames<- assigns a character vector of the document names in a corpus

**Examples**

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

# reassign the document names of the inaugural speech corpus
docnames(inaugCorpus) <- paste("Speech", 1:ndoc(inaugCorpus), sep="")
# query the document names of a dfm
docnames(dfm(inaugTexts[1:5]))
```

---

docvars	<i>get or set for document-level variables</i>
---------	--

---

**Description**

Get or set variables for the documents in a corpus

**Usage**

```
docvars(x, field = NULL)

## S3 method for class 'corpus'
docvars(x, field = NULL)

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

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

## S3 method for class 'corpusSource'
docvars(x, field = NULL)
```

Arguments

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

Value

docvars returns a data.frame of the document-level variables  
docvars<- assigns value to the named field

Examples

```
head(docvars(inaugCorpus))
docvars(inaugCorpus, "President") <- paste("prez", 1:ndoc(inaugCorpus), sep="")
head(docvars(inaugCorpus))
```

---

encodedTexts	<i>encoded texts for testing</i>
--------------	----------------------------------

---

Description

encodedTexts is a 10-element character vector with 10 different encodings

Examples

```
Encoding(encodedTexts)
data.frame(labelled = names(encodedTexts), detected = encoding(encodedTexts)$all)
```

---

encoding	<i>detect the encoding of texts</i>
----------	-------------------------------------

---

Description

Detect the encoding of texts in a character, [corpus](#), or [corpusSource-class](#) object and report on the most likely encoding. Useful in detecting the encoding of input texts, so that a source encoding can be specified when (re)constructing a corpus using [corpus](#).

**Usage**

```
encoding(x, verbose = TRUE, ...)

## S3 method for class 'character'
encoding(x, verbose = TRUE, ...)

## S3 method for class 'corpus'
encoding(x, verbose = TRUE, ...)

## S3 method for class 'corpusSource'
encoding(x, verbose = TRUE, ...)
```

**Arguments**

x	character vector, corpus, or corpusSource object whose texts' encodings will be detected.
verbose	if FALSE, do not print diagnostic report
...	additional arguments passed to <a href="#">stri_enc_detect</a>

**Details**

Based on [stri\\_enc\\_detect](#), which is in turn based on the ICU libraries. See the ICU User Guide, <http://userguide.icu-project.org/conversion/detection>.

**Examples**

```
encoding(encodedTexts)
# show detected value for each text, versus known encoding
data.frame(labelled = names(encodedTexts), detected = encoding(encodedTexts)$all)

encoding(ukimmigTexts)
encoding(inaugCorpus)
encoding(ie2010Corpus)

## Not run: # Russian text, Windows-1251
mytextfile <- textfile("http://www.kenbenoit.net/files/01_er_5.txt", cache = FALSE)
encoding(mytextfile)

## End(Not run)
```

---

exampleString

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


---

**Description**

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

**Format**

character vector with one element

**Examples**

```
tokenize(exampleString, removePunct = TRUE)
```

---

features	<i>extract the feature labels from a dfm</i>
----------	--

---

**Description**

Extract the features from a document-feature matrix, which are stored as the column names of the [dfm](#) object.

**Usage**

```
features(x)

## S3 method for class 'dfm'
features(x)
```

**Arguments**

x                      the object (dfm) whose features will be extracted

**Value**

Character vector of the features

**Examples**

```
features(dfm(inaugTexts))[1:50] # first 50 features (alphabetically sorted)
```

---

head.dfm	<i>Return the first or last part of a dfm</i>
----------	---

---

**Description**

For a [dfm-class](#) object, returns the first or last n documents and first ncol features for inspection.

**Usage**

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

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

**Arguments**

<code>x</code>	a dfm object
<code>n</code>	a single integer. If positive, size for the resulting object: number of first/last documents for the dfm. If negative, all but the <code>n</code> last/first number of documents of <code>x</code> .
<code>nfeature</code>	the number of features to return, where the resulting object will contain the first <code>ncol</code> features
<code>...</code>	unused

**Value**

A [dfm-class](#) class object corresponding to the subset defined by `n` and `ncol`.

A [dfm-class](#) class object corresponding to the subset defined by `n` and `ncol`.

**Examples**

```
myDfm <- dfm(inaugCorpus, ngrams = 2, verbose = FALSE)
head(myDfm)
tail(myDfm)
tail(myDfm, nfeature = 4)
```

---

ie2010Corpus

*Irish budget speeches from 2010*


---

**Description**

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

**Format**

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

**Source**

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

**Examples**

```
summary(ie2010Corpus)
```

---

 inaugCorpus

*A corpus of US presidential inaugural addresses from 1789-2013*


---

### Description

inaugCorpus 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.

inaugTexts is the character vector of US presidential inauguration speeches

### References

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

### Examples

```
# some operations on the inaugural corpus
summary(inaugCorpus)
head(docvars(inaugCorpus), 10)
# working with the character vector only
str(inaugTexts)
head(docvars(inaugCorpus), 10)
mycorpus <- corpus(inaugTexts)
```

---

 kwic

*List key words in context from a text or a corpus of texts.*


---

### Description

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

### Usage

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

```
## S3 method for class 'character'
```

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

```
## S3 method for class 'corpus'
```

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

```
## S3 method for class 'tokenizedTexts'
kwic(x, keywords, window = 5, valuetype = c("glob",
      "regex", "fixed"), case_insensitive = TRUE, ...)

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

## Arguments

<code>x</code>	a text character scalar or a quanteda corpus
<code>keywords</code>	A keyword or phrase consisting of multiple keywords, possibly including punctuation. If a phrase, keywords will be tokenized using the ... options.
<code>window</code>	The number of context words to be displayed around the keyword.
<code>valuetype</code>	how to interpret keyword expressions: "glob" for "glob"-style wildcard expressions; "regex" for regular expressions; or "fixed" for exact matching (entire words, for instance). If "fixed" is used with <code>case_insensitive = TRUE</code> , the text will be lowercased prior to matching.
<code>case_insensitive</code>	match without respect to case if TRUE
<code>...</code>	additional arguments passed to <a href="#">tokenize</a> , for applicable methods

## Value

A kwic object classed data.frame, with the context before (preword), the keyword in its original format (word, preserving case and attached punctuation), the context after (postword), and the index position of the match (position). The rows of the dataframe will be named with the word index position, or the text name and the index position for a corpus object.

## Author(s)

Kenneth Benoit

## Examples

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

kwic(inaugCorpus, "war against")
kwic(inaugCorpus, "war against", valuetype = "regex")
```

---

LBGexample	<i>dfm containing example data from Table 1 of Laver Benoit and Garry (2003)</i>
------------	--

---

## Description

Example data to demonstrate the Wordscores algorithm, from Laver Benoit and Garry (2003)

## Format

A [dfm](#) object with 6 documents and 37 features

## Details

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

## References

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

---

lexdiv	<i>calculate lexical diversity</i>
--------	------------------------------------

---

## Description

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

## Usage

```
lexdiv(x, ...)

## S3 method for class 'dfm'
lexdiv(x, measure = c("TTR", "C", "R", "CTTR", "U", "S",
  "Maas"), log.base = 10, ...)

## S3 method for class 'numeric'
lexdiv(x, measure = c("TTR", "C", "R", "CTTR", "U", "S",
  "Maas"), log.base = 10, ...)
```



## Arguments

x	a <a href="#">document-feature matrix object</a>
...	additional arguments
measure	A character vector defining the measure to calculate.
log.base	A numeric value defining the base of the logarithm.

## Details

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

"TTR": The ordinary *Type-Token Ratio*:

$$TTR = \frac{V}{N}$$

"C": Herdan's  $C$  (Herdan, 1960, as cited in Tweedie & Baayen, 1998; sometimes referred to as *LogTTR*):

$$C = \frac{\log V}{\log N}$$

"R": Guiraud's *Root TTR* (Guiraud, 1954, as cited in Tweedie & Baayen, 1998):

$$R = \frac{V}{\sqrt{N}}$$

"CTTR": Carroll's *Corrected TTR*:

$$CTTR = \frac{V}{\sqrt{2N}}$$

"U": Dugast's *Uber Index* (Dugast, 1978, as cited in Tweedie & Baayen, 1998):

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

"S": Summer's index:

$$S = \frac{\log \log V}{\log \log N}$$

"K": Yule's  $K$  (Yule, 1944, as cited in Tweedie & Baayen, 1998) is calculated by:

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

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

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

$$a^2 = \frac{\log N - \log V}{\log N^2}$$

$$\log V_0 = \frac{\log V}{\sqrt{1 - \frac{\log V^2}{\log N}}}$$

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

### Value

a vector of lexical diversity statistics, each corresponding to an input document

### Note

This implements only the static measures of lexical diversity, not more complex measures based on windows of text such as the Mean Segmental Type-Token Ratio, the Moving-Average Type-Token Ratio (Covington & McFall, 2010), the MLTD or MLTD-MA (Moving-Average Measure of Textual Lexical Diversity) proposed by McCarthy & Jarvis (2010) or Jarvis (no year), or the HD-D version of vocd-D (see McCarthy & Jarvis, 2007). These are available from the package **korRpus**.

### Author(s)

Kenneth Benoit, adapted from the S4 class implementation written by Meik Michalke in the **korRpus** package.

### References

- Covington, M.A. & McFall, J.D. (2010). Cutting the Gordian Knot: The Moving-Average Type-Token Ratio (MATTR). *Journal of Quantitative Linguistics*, 17(2), 94–100.
- Maas, H.-D., (1972). "Über den Zusammenhang zwischen Wortschatzumfang und Länge eines Textes. *Zeitschrift für Literaturwissenschaft und Linguistik*, 2(8), 73–96.
- McCarthy, P.M. & Jarvis, S. (2007). vocd: A theoretical and empirical evaluation. *Language Testing*, 24(4), 459–488.
- McCarthy, P.M. & Jarvis, S. (2010). MTLD, vocd-D, and HD-D: A validation study of sophisticated approaches to lexical diversity assessment. *Behaviour Research Methods*, 42(2), 381–392.
- Michalke, Meik. (2014) *koRpus: An R Package for Text Analysis*. Version 0.05-5. <http://reaktanz.de/?c=hacking&s=koRpus>
- Tweedie, F.J. & Baayen, R.H. (1998). How Variable May a Constant Be? Measures of Lexical Richness in Perspective. *Computers and the Humanities*, 32(5), 323–352.

### Examples

```
mydfm <- dfm(subset(inaugCorpus, Year>1980))
mydfmSW <- dfm(subset(inaugCorpus, Year>1980), ignoredFeatures=stopwords("english"))
results <- data.frame(TTR = lexdiv(mydfm, "TTR"),
                      CTTR = lexdiv(mydfm, "CTTR"),
                      U = lexdiv(mydfm, "U"),
                      TTRs = lexdiv(mydfmSW, "TTR"),
                      CTTRs = lexdiv(mydfmSW, "CTTR"),
                      Us = lexdiv(mydfmSW, "U"))
```

```
results
cor(results)
t(lexdiv(mydfmSW, "Maas"))
```

---

metacorporus	<i>get or set corpus metadata</i>
--------------	-----------------------------------

---

## Description

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

## Usage

```
metacorporus(x, field = NULL)

## S3 method for class 'corpus'
metacorporus(x, field = NULL)

metacorporus(x, field) <- value
```

## Arguments

x	A quanteda corpus object
field	Metadata field name(s). If NULL (default), return all metadata names.
value	new value of the corpus metadata field

## Value

For metacorporus, a list of the metadata fields in the corpus. If a list is not what you wanted, you can wrap the results in [unlist](#), but this will remove any metadata field that is set to NULL.

For metacorporus <-, the corpus with the updated metadata.

## Examples

```
metacorporus(inaugCorpus)
metacorporus(inaugCorpus, "source")
metacorporus(inaugCorpus, "citation") <- "Presidential Speeches Online Project (2014)."
```

---

metadoc	<i>get or set document-level meta-data</i>
---------	--

---

## Description

Get or set the document-level meta-data, including reserved fields for language and corpus.

## Usage

```
metadoc(x, field = NULL)

## S3 method for class 'corpus'
metadoc(x, field = NULL)

metadoc(x, field = NULL) <- value

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

## Arguments

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

## Value

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

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

## Note

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

## Examples

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

---

ndoc	<i>get the number of documents or features</i>
------	--

---

## Description

ndoc returns the number of documents or features in a quanteda object, which can be a corpus, dfm, or tokenized texts.

nfeature is an alias for ntype when applied to dfm objects. For a corpus or set of texts, "features" are only defined through tokenization, so you need to use [ntoken](#) to count these.

## Usage

```
ndoc(x)

## S3 method for class 'corpus'
ndoc(x)

## S3 method for class 'dfm'
ndoc(x)

nfeature(x)

## S3 method for class 'corpus'
nfeature(x)

## S3 method for class 'dfm'
nfeature(x)
```

## Arguments

x                      a corpus or dfm object

## Value

an integer (count) of the number of documents or features in the corpus or dfm

## Examples

```
ndoc(subset(inaugCorpus, Year>1980))
ndoc(dfm(subset(inaugCorpus, Year>1980), verbose=FALSE))
nfeature(dfm(inaugCorpus))
nfeature(trim(dfm(inaugCorpus), minDoc=5, minCount=10))
```

ngrams

*Create ngrams and skipgrams***Description**

Create a set of ngrams (tokens in sequence) from character vectors or tokenized text objects, with an optional skip argument to form skipgrams. Both the ngram length and the skip lengths take vectors of arguments to form multiple lengths or skips in one pass. `ngrams()` is implemented in C++ for efficiency.

**Usage**

```
ngrams(x, ...)

## S3 method for class 'character'
ngrams(x, n = 2L, skip = 0L, concatenator = "_", ...)

## S3 method for class 'tokenizedTexts'
ngrams(x, n = 2L, skip = 0L, concatenator = "_",
      ...)

skipgrams(x, ...)

## S3 method for class 'character'
skipgrams(x, n, skip, concatenator = "_", ...)

## S3 method for class 'tokenizedTexts'
skipgrams(x, n, skip, concatenator = "_", ...)
```

**Arguments**

<code>x</code>	a tokenizedText object or a character vector of tokens
<code>...</code>	not used
<code>n</code>	integer vector specifying the number of elements to be concatenated in each ngram
<code>skip</code>	integer vector specifying the adjacency skip size for tokens forming the ngrams, default is 0 for only immediately neighbouring words. For skipgrams, skip can be a vector of integers, as the "classic" approach to forming skip-grams is to set <code>skip = k</code> 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 <code>skip = 0:4</code> 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).
<code>concatenator</code>	character for combining words, default is <code>_</code> (underscore) character

## Details

Normally, `ngrams` will be called through `tokenize`, but these functions are also exported in case a user wants to perform lower-level ngram construction on tokenized texts.

`skipgrams` is a wrapper to `ngrams` that requires arguments to be supplied for both `n` and `skip`. For  $k$ -skip skipgrams, set `skip` to  $0:k$ , in order to conform to the definition of skip-grams found in Guthrie et al (2006): A  $k$  skip-gram is an ngram which is a superset of all ngrams and each  $(k - i)$  skipgram until  $(k - i) == 0$  (which includes 0 skip-grams).

## Value

a `tokenizedTexts` object consisting a list of character vectors of ngrams, one list element per text, or a character vector if called on a simple character vector

## Author(s)

Kohei Watanabe and Ken Benoit

## References

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

## Examples

```
# ngrams
ngrams(LETTERS, n = 2)
ngrams(LETTERS, n = 2, skip = 1)
ngrams(LETTERS, n = 2, skip = 0:1)
ngrams(LETTERS, n = 1:2)
ngrams(LETTERS, n = c(2,3), skip = 0:1)

tokens <- tokenize("the quick brown fox jumped over the lazy dog.",
  removePunct = TRUE, simplify = TRUE)
ngrams(tokens, n = 1:3)
ngrams(tokens, n = c(2,4), concatenator = " ")
ngrams(tokens, n = c(2,4), skip = 1, concatenator = " ")

# skipgrams
tokens <- tokenize(toLower("Insurgents killed in ongoing fighting."),
  removePunct = TRUE, simplify = TRUE)
skipgrams(tokens, n = 2, skip = 0:1, concatenator = " ")
skipgrams(tokens, n = 2, skip = 0:2, concatenator = " ")
skipgrams(tokens, n = 3, skip = 0:2, concatenator = " ")
```

---

nsentence	<i>count the number of sentences</i>
-----------	--------------------------------------

---

## Description

Return the count of sentences in a corpus or character.

## Usage

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

## Arguments

x	texts or corpus whose sentences will be counted
...	additional arguments passed to <a href="#">tokenize</a>

## Value

scalar count(s) of the total sentences per text

## Note

‘nsentence()’ relies on the boundaries definitions in the **stringi** package (see [stri\\_opts\\_brkiter](#)). It does not count sentences correctly if the text has been transformed to lower case, and for this reason ‘nsentence()’ will stop with an error 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.")  
nsentence(txt)
```



---

**ntoken***count the number of tokens or types*

---

### Description

Return the count of tokens (total features) or types (unique features) in a text, corpus, or dfm. "tokens" here means all words, not unique words, and these are not cleaned prior to counting.

### Usage

```
ntoken(x, ...)  
  
ntype(x, ...)  
  
## S3 method for class 'corpus'  
ntoken(x, ...)  
  
## S3 method for class 'corpus'  
ntype(x, ...)  
  
## S3 method for class 'character'  
ntoken(x, ...)  
  
## S3 method for class 'tokenizedTexts'  
ntoken(x, ...)  
  
## S3 method for class 'character'  
ntype(x, ...)  
  
## S3 method for class 'dfm'  
ntoken(x, ...)  
  
## S3 method for class 'dfm'  
ntype(x, ...)  
  
## S3 method for class 'tokenizedTexts'  
ntype(x, ...)
```

### Arguments

x	texts or corpus whose tokens or types will be counted
...	additional arguments passed to <a href="#">tokenize</a>

### Value

scalar count of the total tokens or types

**Note**

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

**Examples**

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

# with some real texts
ntoken(subset(inaugCorpus, Year<1806, removePunct = TRUE))
ntype(subset(inaugCorpus, Year<1806, removePunct = TRUE))
ntoken(dfm(subset(inaugCorpus, Year<1800)))
ntype(dfm(subset(inaugCorpus, Year<1800)))
```

---

phrasetotoken	<i>convert phrases into single tokens</i>
---------------	---

---

**Description**

Replace multi-word phrases in text(s) with a compound version of the phrases concatenated with concatenator (by default, the "\_" character) to form a single token. This prevents tokenization of the phrases during subsequent processing by eliminating the whitespace delimiter.

**Usage**

```
phrasetotoken(object, phrases, concatenator = "_")

## S4 method for signature 'character,dictionary'
phrasetotoken(object, phrases,
  concatenator = "_")

phrasetotoken.corpus(object, phrases, concatenator = "_")

## S4 method for signature 'character,collocations'
phrasetotoken(object, phrases,
  concatenator = "_")
```

**Arguments**

object	source texts, a character or character vector
phrases	a <a href="#">dictionary</a> object that contains some phrases, defined as multiple words delimited by whitespace, up to 9 words long; or a quanteda collocation object created by <a href="#">collocations</a>
concatenator	the concatenation character that will connect the words making up the multi-word phrases. The default <code>_</code> is highly recommended since it will not be removed during normal cleaning and tokenization (while nearly all other punctuation characters, at least those in the POSIX class <code>[:punct:]</code> ) will be removed.

**Value**

character or character vector of texts with phrases replaced by compound "words" joined by the concatenator

**Author(s)**

Kenneth Benoit

**Examples**

```
mytexts <- c("The new law included a capital gains tax, and an inheritance tax.",
            "New York City has raised a taxes: an income tax and a sales tax.")
mydict <- dictionary(list(tax=c("tax", "income tax", "capital gains tax", "inheritance tax")))
(cw <- phrasetotoken(mytexts, mydict))
dfm(cw, verbose=FALSE)

# when used as a dictionary for dfm creation
mydfm2 <- dfm(cw, dictionary=lapply(mydict, function(x) gsub(" ", "_", x)))
mydfm2
# to pick up "taxes" in the second text, set dictionary_regex=TRUE
mydfm3 <- dfm(cw, dictionary=lapply(mydict, phrasetotoken, mydict),
              dictionary_regex=TRUE)
mydfm3
## one more token counted for "tax" than before
```

---

plot.dfm

*plot features as a wordcloud*


---

**Description**

The default plot method for a [dfm](#) object. Produces a wordcloud plot for the features of the dfm, where the feature labels are plotted with their sizes proportional to their numerical values in the dfm. When `comparison = TRUE`, it plots comparison word clouds by document.

**Usage**

```
## S3 method for class 'dfm'
plot(x, comparison = FALSE, ...)
```

## Arguments

x	a dfm object
comparison	if TRUE, plot a <a href="#">comparison.cloud</a> instead of a simple wordcloud, one grouping per document
...	additional parameters passed to <a href="#">wordcloud</a> or to <a href="#">text</a> (and <a href="#">strheight</a> , <a href="#">strwidth</a> )

## Details

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

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

## See Also

[wordcloud](#), [comparison.cloud](#)

## Examples

```
# plot the features (without stopwords) from Obama's two inaugural addresses
mydfm <- dfm(subset(inaugCorpus, President=="Obama"), verbose = FALSE,
             ignoredFeatures = stopwords("english"))
plot(mydfm)

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

## Not run: # comparison plot of Irish government vs opposition
govtopp <- factor(ifelse(ie2010Corpus[, "party"] %in% c("FF", "Green"), "Govt", "Opp"))
govtoppDfm <- dfm(ie2010Corpus, groups = govtopp, verbose = FALSE)
plot(tfidf(govtoppDfm), comparison = TRUE)
# compare to non-tf-idf version
plot(sideDfm, comparison = TRUE)
## End(Not run)
```

---

plot.kwic

*plot a dispersion plot of key word(s)*

---

## Description

Plots a dispersion or "x-ray" plot of a selected word pattern across a text.

**Usage**

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

**Arguments**

x                    a [kwic](#) class object

...                  additional arguments passed to [plot](#)

**Note**

Currently works only for a a single document

---

print.dfm	<i>print a dfm object</i>
-----------	---------------------------

---

**Description**

print methods for document-feature matrices

**Usage**

```
## S4 method for signature 'dfm'
print(x, show.values = FALSE, show.settings = FALSE,
      show.summary = TRUE, ndoc = 20L, nfeature = 20L, ...)

## S4 method for signature 'dfmSparse'
print(x, show.values = FALSE, show.settings = FALSE,
      show.summary = TRUE, ndoc = 20L, nfeature = 20L, ...)

## S4 method for signature 'dfmDense'
print(x, show.values = FALSE, show.settings = FALSE,
      show.summary = TRUE, ndoc = 20L, nfeature = 20L, ...)

## S4 method for signature 'dfmSparse'
show(object)

## S4 method for signature 'dfmDense'
show(object)

## S3 method for class 'dfm'
print(x, show.values = FALSE, show.settings = FALSE,
      show.summary = TRUE, ndoc = 20L, nfeature = 20L, ...)
```

**Arguments**

<code>x</code>	the dfm to be printed
<code>show.values</code>	print the dfm as a matrix or array (if resampled).
<code>show.settings</code>	print the settings used to create the dfm. See <a href="#">settings</a> .
<code>show.summary</code>	print a brief summary indicating the number of documents and features
<code>ndoc</code>	max number of documents to print
<code>nfeature</code>	max number of features to print
<code>...</code>	further arguments passed to or from other methods
<code>object</code>	the item to be printed

---

`print.tokenizedTexts`    *print a tokenizedTexts objects*

---

**Description**

print method for a [tokenizedText](#) object

**Usage**

```
## S3 method for class 'tokenizedTexts'
print(x, ...)
```

**Arguments**

<code>x</code>	a tokenizedText object created by <a href="#">tokenize</a>
<code>...</code>	further arguments passed to base print method

---

`readability`            *calculate readability*

---

**Description**

Calculate the readability of text(s).

**Usage**

```
readability(x, measure, drop = TRUE)

## S3 method for class 'corpus'
readability(x, measure, drop = TRUE)

## S3 method for class 'character'
readability(x, measure = c("all", "ARI", "ARI.simple",
  "Bormuth", "Bormuth.GP", "Coleman", "Coleman.C2", "Coleman.Liau",
  "Coleman.Liau.grade", "Coleman.short", "Dale.Chall", "Dale.Chall.old",
  "Dale.Chall.PSK", "Danielson.Bryan", "Danielson.Bryan.2", "Dickes.Steiwer",
  "DRP", "ELF", "Farr.Jenkins.Paterson", "Flesch", "Flesch.PSK",
  "Flesch.Kincaid", "FOG", "FOG.PSK", "FOG.NRI", "FORCAST", "FORCAST.RGL",
  "Fucks", "Linsear.Write", "LIW", "nWS", "nWS.2", "nWS.3", "nWS.4", "RIX",
  "Scrabble", "SMOG", "SMOG.C", "SMOG.simple", "SMOG.de", "Spache",
  "Spache.old", "Strain", "Traenkle.Bailer", "Traenkle.Bailer.2",
  "Wheeler.Smith", "meanSentenceLength", "meanWordSyllables"), drop = TRUE)
```

**Arguments**

x	a <a href="#">corpus</a> object or character vector
measure	character vector defining the readability measure to calculate
drop	if TRUE, the result is returned as a numeric vector if only a single measure is requested; otherwise, a data.frame is returned with each column consisting of a requested measure.

**Value**

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

**Author(s)**

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

**Examples**

```
readability(inaugCorpus, measure = "Flesch.Kincaid")
txt <- c("Readability zero one. Ten, Eleven.", "The cat in a dilapidated tophat.")
readability(txt, "Flesch.Kincaid")
readability(txt, "Flesch.Kincaid", drop = FALSE)
readability(txt, c("FOG", "FOG.PSK", "FOG.NRI"))
inaugReadability <- readability(inaugCorpus, "all")
round(cor(inaugReadability), 3)
```

---

removeFeatures	<i>remove features from an object</i>
----------------	---------------------------------------

---

## Description

This function removes features from a variety of objects, such as text, a dfm, or a list of collocations. The most common usage for removeFeatures will be to eliminate stop words from a text or text-based object. Some commonly used built-in stop words can be accessed through [stopwords](#).

## Usage

```
removeFeatures(x, stopwords = NULL, verbose = TRUE, ...)

## S3 method for class 'tokenizedTexts'
removeFeatures(x, stopwords = NULL, verbose = TRUE,
  ...)

## S3 method for class 'dfm'
removeFeatures(x, stopwords = NULL, verbose = TRUE, ...)

## S3 method for class 'collocations'
removeFeatures(x, stopwords = NULL, verbose = TRUE,
  pos = c(1, 2, 3), ...)

stopwordsRemove(x, stopwords = NULL, verbose = TRUE)
```

## Arguments

x	object from which stopwords will be removed
stopwords	character vector of features to remove. Now requires an explicit list to be supplied, for instance stopwords("english").
verbose	if TRUE print message about how many features were removed
...	additional arguments for some methods (such as pos for <a href="#">collocations</a> )
pos	indexes of word position if called on collocations: remove if word pos is a stopwords

## Details

Because we believe the user should take full responsibility for any features that are removed, we do not provide a default list. Use [stopwords](#) instead.

## Value

an object with stopwords removed



**Author(s)**

Kenneth Benoit

**See Also**

[stopwords](#)

**Examples**

```
## examples for character objects
someText <- tokenize(c(text1 = "Here's some text containing words we want to remove."))
removeFeatures(someText, stopwords("english"))
removeFeatures(someText, stopwords("SMART"))
removeFeatures(someText, c("some", "want"))
someText <- tokenize(c(text1 = "Here's some text containing words we want to remove.",
                      text2 = "A second sentence with a few stopwords."))
removeFeatures(someText, stopwords("english"))

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

itText <- tokenize("Ecco alcuni di testo contenente le parole che vogliamo rimuovere.",
                  removePunct = TRUE)
removeFeatures(itText, stopwords("italian"), case_insensitive = TRUE)

## example for dfm objects
mydfm <- dfm(ukimmigTexts, verbose=FALSE)
removeFeatures(mydfm, stopwords("english"))

## example for collocations
(myCollocs <- collocations(inaugTexts[1:3], top=20))
removeFeatures(myCollocs, stopwords("english", verbose=FALSE))
```

---

sample.corpus

*Randomly sample documents or features*

---

**Description**

Takes a random sample or documents or features of the specified size from a corpus or document-feature matrix, with or without replacement

**Usage**

```
## S3 method for class 'corpus'
sample(x, size = ndoc(x), replace = FALSE, prob = NULL,
  ...)

sample(x, size, replace = FALSE, prob = NULL, ...)

## Default S3 method:
sample(x, size, replace = FALSE, prob = NULL, ...)

## S3 method for class 'dfm'
sample(x, size = ndoc(x), replace = FALSE, prob = NULL,
  what = c("documents", "features"), ...)
```

**Arguments**

x	a corpus or dfm object whose documents or features will be sampled
size	a positive number, the number of documents to select
replace	Should sampling be with replacement?
prob	A vector of probability weights for obtaining the elements of the vector being sampled.
...	unused <a href="#">sample</a> , which is not defined as a generic method in the <b>base</b> package.
what	dimension (of a <a href="#">dfm</a> ) to sample: can be documents or features

**Value**

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

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

**See Also**

[sample](#)

[sample](#)

**Examples**

```
# sampling from a corpus
summary(sample(inaugCorpus, 5))
summary(sample(inaugCorpus, 10, replace=TRUE))
# sampling from a dfm
myDfm <- dfm(inaugTexts[1:10], verbose = FALSE)
sample(myDfm)[, 1:10]
sample(myDfm, replace = TRUE)[, 1:10]
sample(myDfm, what = "features")[1:10, ]
```

---

scrabble	<i>compute the Scrabble letter values of text</i>
----------	---

---

**Description**

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

**Usage**

```
scrabble(x, FUN = sum)

## S3 method for class 'character'
scrabble(x, FUN = sum)
```

**Arguments**

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

**Value**

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

**Note**

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

**Author(s)**

Kenneth Benoit

**Examples**

```
scrabble(c("muzjiks", "excellency"))
scrabble(inaugTexts[1:5], mean)
```

segment

*segment texts into component elements***Description**

Segment text(s) into tokens, sentences, paragraphs, or other sections. `segment` works on a character vector or corpus object, and allows the delimiters to be defined. See details.

**Usage**

```
segment(x, ...)
```

```
## S3 method for class 'character'
segment(x, what = c("tokens", "sentences", "paragraphs",
  "tags", "other"), delimiter = ifelse(what == "tokens", " ", ifelse(what ==
  "sentences", "[.!?;:]", ifelse(what == "paragraphs", "\\n{2}", ifelse(what
  == "tags", "##\\w+\\b", NULL))))), perl = FALSE, ...)
```

```
## S3 method for class 'corpus'
segment(x, what = c("tokens", "sentences", "paragraphs",
  "tags", "other"), delimiter = ifelse(what == "tokens", " ", ifelse(what ==
  "sentences", "[.!?;:]", ifelse(what == "paragraphs", "\\n{2}", ifelse(what
  == "tags", "##\\w+\\b", NULL))))), perl = FALSE, ...)
```

**Arguments**

<code>x</code>	text or corpus object to be segmented
<code>...</code>	provides additional arguments passed to <code>tokenize</code> , if <code>what = "tokens"</code> is used
<code>what</code>	unit of segmentation. Current options are tokens, sentences, paragraphs, and other. Segmenting on other allows segmentation of a text on any user-defined value, and must be accompanied by the <code>delimiter</code> argument.
<code>delimiter</code>	delimiter defined as a <a href="#">regex</a> for segmentation. Each type has its own default, except other, which requires a value to be specified.
<code>perl</code>	logical. Should Perl-compatible regular expressions be used?

**Details**

Tokens are delimited by Separators. For sentences, the delimiter can be defined by the user. The default for sentences includes `.`, `!`, `?`, plus `;` and `:`.

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

**Value**

A list of segmented texts, with each element of the list corresponding to one of the original texts.

**Note**

Does not currently record document segments if segmenting a multi-text corpus into smaller units. For this, use [changeunits](#) instead.

**Examples**

```
# same as tokenize()
identical(tokenize(ukimmigTexts, lower=FALSE), segment(ukimmigTexts, lower=FALSE))

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

# segment a text into sentences
segmentedChar <- segment(ukimmigTexts, "sentences")
segmentedChar[2]
testCorpus <- corpus("##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.")
testCorpusSeg <- segment(testCorpus, "tags")
summary(testCorpusSeg)
texts(testCorpusSeg)
# segment a corpus into sentences
segmentedCorpus <- segment(corpus(ukimmigTexts), "sentences")
identical(ndoc(segmentedCorpus), length(unlist(segmentedChar)))
```

---

selectFeatures	<i>select features from an object</i>
----------------	---------------------------------------

---

**Description**

This function selects or discards features from a dfm.vocabulary of objects, such as tokenized texts, a dfm, or a list of collocations. The most common usage for removeFeatures will be to eliminate stop words from a text or text-based object, or to select only features from a list of regular expression.

**Usage**

```
selectFeatures(x, features, ...)

## S3 method for class 'dfm'
selectFeatures(x, features = NULL, selection = c("keep",
        "remove"), valuetype = c("glob", "regex", "fixed"),
        case_insensitive = TRUE, verbose = TRUE, ...)
```

## Arguments

x	object whose features will be selected
features	one of: a character vector of features to be selected, a <a href="#">dfm</a> whose features will be used for selection, or a dictionary class object whose values (not keys) will provide the features to be selected. For <a href="#">dfm</a> objects, see details in the Value section below.
...	supplementary arguments passed to the underlying functions in <a href="#">stri_detect_regex</a> . (This is how <code>case_insensitive</code> is passed, but you may wish to pass others.)
selection	whether to keep or remove the features
valuetype	how to interpret feature vector: fixed for words as is; "regex" for regular expressions; or "glob" for "glob"-style wildcard
case_insensitive	ignore the case of dictionary values if TRUE
verbose	if TRUE print message about how many features were removed

## Value

A dfm after the feature selection has been applied.

When `features` is a [dfm-class](#) object, then the returned object will be identical in its feature set to the dfm supplied as the `features` argument. This means that any features in `x` not in `features` will be discarded, and that any features in found in the dfm supplied as `features` but not found in `x` will be added with all zero counts. This is useful when you have trained a model on one dfm, and need to project this onto a test set whose features must be identical.

## Note

This function selects features based on their labels. To select features based on the values of a the document-feature matrix, use [trim](#).

## See Also

[removeFeatures](#), [trim](#)

## Examples

```
myDfm <- dfm(c("My Christmas was ruined by your opposition tax plan.",
              "Does the United_States or Sweden have more progressive taxation?"),
            toLower = FALSE, verbose = FALSE)
mydict <- dictionary(list(countries = c("United_States", "Sweden", "France"),
                          wordsEndingInY = c("by", "my"),
                          notintext = "blahblah"))
selectFeatures(myDfm, mydict)
selectFeatures(myDfm, mydict, case_insensitive = FALSE)
selectFeatures(myDfm, c("s$", ".y"), "keep")
selectFeatures(myDfm, c("s$", ".y"), "keep", valuetype = "regex")
selectFeatures(myDfm, c("s$", ".y"), "remove", valuetype = "regex")
selectFeatures(myDfm, stopwords("english"), "keep", valuetype = "fixed")
selectFeatures(myDfm, stopwords("english"), "remove", valuetype = "fixed")
```

```
# selecting on a dfm
textVec1 <- c("This is text one.", "This, the second text.", "Here: the third text.")
textVec2 <- c("Here are new words.", "New words in this text.")
features(dfm1 <- dfm(textVec1))
features(dfm2a <- dfm(textVec2))
(dfm2b <- selectFeatures(dfm2a, dfm1))
identical(features(dfm1), features(dfm2b))
```

---

settings

*Get or set the corpus settings*


---

## Description

Get or set the corpus settings

Get or set various settings in the corpus for the treatment of texts, such as rules for stemming, stopwords, collocations, etc.

Get the settings from a which a [dfm](#) was created

## Usage

```
settings(x, ...)

## Default S3 method:
settings(x = NULL, ...)

## S3 method for class 'corpus'
settings(x, field = NULL, ...)

settings(x, field) <- value

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

## S3 method for class 'settings'
print(x, ...)
```

## Arguments

x	object from/to which settings are queried or applied
...	additional arguments
field	string containing the name of the setting to be set or queried settings(x) query the corps settings settings(x, field) <- update the corpus settings for field
value	new setting value

**Details**

Calling `settings()` with no arguments returns a list of system default settings.

**Examples**

```
settings(inaugCorpus, "stopwords")
(tempdfm <- dfm(subset(inaugCorpus, Year>1980), verbose=FALSE))
(tempdfmSW <- dfm(subset(inaugCorpus, Year>1980),
  ignoredFeatures=stopwords("english"), verbose=FALSE))
settings(inaugCorpus, "stopwords") <- TRUE
tempdfm <- dfm(inaugCorpus, stem=TRUE, verbose=FALSE)
settings(tempdfm)
```

---

`show, dictionary-method`

*print a dictionary object*

---

**Description**

Print/show method for dictionary objects.

**Usage**

```
## S4 method for signature 'dictionary'
show(object)
```

**Arguments**

`object`                      the dictionary to be printed

---

`similarity`

*compute similarities between documents and/or features*

---

**Description**

Compute similarities between documents and/or features from a [dfm](#). Uses the similarity measures defined in [simil](#). See [pr\\_DB](#) for available distance measures, or how to create your own.

**Usage**

```
similarity(x, selection, n = 10, margin = c("features", "documents"),
  method = "correlation", sort = TRUE, normalize = TRUE, digits = 4)

## S4 method for signature 'dfm,index'
similarity(x, selection, n = 10,
  margin = c("features", "documents"), method = "correlation",
  sort = TRUE, normalize = TRUE, digits = 4)
```



**Arguments**

<code>x</code>	a <code>dfm</code> object
<code>selection</code>	character or character vector of document names or feature labels from the <code>dfm</code>
<code>n</code>	the top <code>n</code> most similar items will be returned, sorted in descending order. If <code>n</code> is <code>NULL</code> , return all items.
<code>margin</code>	identifies the margin of the <code>dfm</code> on which similarity will be computed: features for word/term features or documents for documents.
<code>method</code>	a valid method for computing similarity from <code>pr_DB</code>
<code>sort</code>	sort results in descending order if <code>TRUE</code>
<code>normalize</code>	if <code>TRUE</code> , normalize the <code>dfm</code> by term frequency within document (so that the <code>dfm</code> values will be relative term frequency within each document)
<code>digits</code>	digits for rounding results

**Value**

a named list of the selection labels, with a sorted named vector of similarity measures.

**Note**

The method for computing feature similarities can be quite slow when there are large numbers of feature types. Future implementations will hopefully speed this up.

**Examples**

```
# create a dfm from inaugural addresses from Reagan onwards
presDfm <- dfm(subset(inaugCorpus, Year>1980), ignoredFeatures=stopwords("english"),
               stem=TRUE)

# compute some document similarities
similarity(presDfm, "1985-Reagan", n=5, margin="documents")
similarity(presDfm, c("2009-Obama", "2013-Obama"), n=5, margin="documents")
similarity(presDfm, c("2009-Obama", "2013-Obama"), n=NULL, margin="documents")
similarity(presDfm, c("2009-Obama", "2013-Obama"), n=NULL, margin="documents", method="cosine")
similarity(presDfm, "2005-Bush", n=NULL, margin="documents", method="eJaccard", sort=FALSE)

# compute some term similarities
similarity(presDfm, c("fair", "health", "terror"), method="cosine")

## Not run:

# compare to tm
require(tm)
data("crude")
crude <- tm_map(crude, content_transformer(tolower))
crude <- tm_map(crude, removePunctuation)
crude <- tm_map(crude, removeNumbers)
crude <- tm_map(crude, stemDocument)
tdm <- TermDocumentMatrix(crude)
findAssocs(tdm, c("oil", "opec", "xyz"), c(0.75, 0.82, 0.1))
```

```
# in quanteda
crudeDfm <- dfm(corpus(crude))
similarity(crudeDfm, c("oil", "opec", "xyz"), normalize=FALSE, digits=2)

## End(Not run)
```

---

sort.dfm	<i>sort a dfm by one or more margins</i>
----------	--

---

## Description

Sorts a [dfm](#) by frequency of total features, total features in documents, or both

## Usage

```
## S3 method for class 'dfm'
sort(x, decreasing = TRUE, margin = c("features", "docs",
  "both"), ...)
```

## Arguments

x	Document-feature matrix created by <a href="#">dfm</a>
decreasing	TRUE (default) if sort will be in descending order, otherwise sort in increasing order
margin	which margin to sort on features to sort by frequency of features, docs to sort by total feature counts in documents, and both to sort by both
...	additional arguments passed to base method <code>sort.int</code>

## Value

A sorted [dfm](#) matrix object

## Author(s)

Ken Benoit

## Examples

```
dtm <- dfm(inaugCorpus)
dtm[1:10, 1:5]
dtm <- sort(dtm)
sort(dtm)[1:10, 1:5]
sort(dtm, TRUE, "both")[1:10, 1:5] # note that the decreasing=TRUE argument
# must be second, because of the order of the
# formals in the generic method of sort()
```

---

`stopwords`*access built-in stopwords*

---

### Description

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

### Usage

```
stopwords(kind = "english", verbose = FALSE)
```

```
stopwordsGet(kind = "english")
```

### Arguments

<code>kind</code>	The pre-set kind of stopwords (as a character string). Allowed values are english, SMART, danish, french, hungarian, norwegian, russian, swedish, catalan, dutch, finnish, german, italian, portuguese, spanish, arabic
<code>verbose</code>	if FALSE, suppress the annoying warning note

### Details

The stopword list are SMART English stopwords from the SMART information retrieval system (obtained from <http://jmlr.csail.mit.edu/papers/volume5/lewis04a/all-smart-stop-list/english.stop>) and a set of stopword lists from the Snowball stemmer project in different languages (obtained from [http://svn.tartarus.org/snowball/trunk/website/algorithms/\\*/stop.txt](http://svn.tartarus.org/snowball/trunk/website/algorithms/*/stop.txt)). Supported languages are arabic, danish, dutch, english, finnish, french, german, hungarian, italian, norwegian, portuguese, russian, spanish, and swedish. Language names are case sensitive.

### Value

a character vector of stopwords

### A note of caution

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

### Examples

```
stopwords("english")[1:5]  
stopwords("italian")[1:5]  
stopwords("arabic")[1:5]
```

---

subset.corpus	<i>extract a subset of a corpus</i>
---------------	-------------------------------------

---

### Description

Returns subsets of a corpus that meet certain conditions, including direct logical operations on docvars (document-level variables). Works just like the normal subset command but for corpus objects.

### Usage

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

### Arguments

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

### Value

corpus object

### See Also

[select](#)

### Examples

```
summary(subset(inaugCorpus, Year>1980))
summary(subset(inaugCorpus, Year>1930 & President=="Roosevelt", select=Year))
```

---

summary.corpus	<i>summarize a corpus or a vector of texts</i>
----------------	--

---

### Description

Displays information about a corpus or vector of texts. For a corpus, this includes attributes and metadata such as date of number of texts, creation and source. For texts, prints to the console a description of the texts, including number of types, tokens, and sentences.

**Usage**

```
## S3 method for class 'corpus'
summary(object, n = 100, verbose = TRUE,
        showmeta = FALSE, ...)

## S3 method for class 'character'
summary(object, verbose = TRUE, ...)

describeTexts(object, verbose = TRUE, ...)
```

**Arguments**

<code>object</code>	corpus or texts to be summarized
<code>n</code>	maximum number of texts to describe, default=100
<code>verbose</code>	set to FALSE to turn off printed output, for instance if you simply want to assign the output to a <code>data.frame</code>
<code>showmeta</code>	for a corpus, set to TRUE to include document-level meta-data
<code>...</code>	unused here

**Examples**

```
# summarize corpus information
summary(inaugCorpus)
summary(inaugCorpus, n=10)
mycorpus <- corpus(ukimmigTexts, docvars=data.frame(party=names(ukimmigTexts)), enc="UTF-8")
summary(mycorpus, showmeta=TRUE) # show the meta-data
mysummary <- summary(mycorpus, verbose=FALSE) # (quietly) assign the results
mysummary$Types / mysummary$Tokens          # crude type-token ratio
#
# summarize texts
summary(c("testing this text", "and this one"))
summary(ukimmigTexts)
myTextSummaryDF <- summary(ukimmigTexts, verbose=FALSE)
```

---

syllables

*count syllables in a text*


---

**Description**

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

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

**Usage**

```
syllables(x, ...)

## S3 method for class 'character'
syllables(x, syllableDict = quanteda::englishSyllables,
  ...)

## S3 method for class 'tokenizedTexts'
syllables(x,
  syllableDict = quanteda::englishSyllables, ...)
```

**Arguments**

<code>x</code>	character vector or tokenizedText-class object whose syllables will be counted
<code>...</code>	additional arguments passed to tokenize
<code>syllableDict</code>	optional named integer vector of syllable counts where the names are lower case tokens. When set to NULL (default), then the function will use the quanteda data object englishSyllables, an English pronunciation dictionary from CMU.

**Value**

If `x` is a character vector, a named numeric vector of the counts of the syllables in each text, without tokenization. If `x` consists of (a list of) tokenized texts, then return a list of syllable counts corresponding to the tokenized texts.

**Note**

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

**Source**

englishSyllables is built from the freely available CMU pronunciation dictionary at <http://www.speech.cs.cmu.edu/cg>

**Examples**

```
syllables("This is an example sentence.")
syllables(tokenize("This is an example sentence.", simplify=TRUE))
myTexts <- c(text1 = "Text one.",
             text2 = "Superduper text number two.",
             text3 = "One more for the road.")
syllables(myTexts)
syllables(tokenize(myTexts, removePunct = TRUE))
syllables("supercalifragilisticexpialidocious")
```

---

textfile	<i>read a text corpus source from a file</i>
----------	--

---

## Description

Read a text corpus from a source file, where the single file will consist of a set of texts in columns and document variables and document-level meta-data in additional columns. For spreadsheet-like files, the first row must be a header.

## Usage

```
textfile(file, textField, encodingFrom = NULL, encodingTo = "UTF-8",
  cache = FALSE, docvarsfrom = c("filenames"), dvsep = "_",
  docvarnames = NULL, ...)

## S4 method for signature
## 'character,index,missing,missing,ANY,missing,missing,missing'
textfile(file,
  textField, encodingFrom = NULL, encodingTo = "UTF-8", cache = FALSE,
  docvarsfrom = c("filenames"), dvsep = "_", docvarnames = NULL, ...)

## S4 method for signature
## 'character,missing,ANY,ANY,ANY,missing,missing,missing'
textfile(file,
  textField, encodingFrom = NULL, encodingTo = "UTF-8", cache = FALSE,
  docvarsfrom = c("filenames"), dvsep = "_", docvarnames = NULL, ...)

## S4 method for signature
## 'character,missing,missing,missing,ANY,character,ANY,ANY'
textfile(file,
  textField, encodingFrom = NULL, encodingTo = "UTF-8", cache = FALSE,
  docvarsfrom = c("filenames"), dvsep = "_", docvarnames = NULL, ...)
```

## Arguments

file	the complete filename to be read. Currently available file types are:
	txt plain text files
	json data in JavaScript Object Notation, consisting of the texts and additional document-level variables and document-level meta-data. The text key must be identified by specifying a textField value.
	csv comma separated value data, consisting of the texts and additional document-level variables and document-level meta-data. The text file must be identified by specifying a textField value.

	<p>tab, tsv tab-separated value data, consisting of the texts and additional document-level variables and document-level meta-data. The text file must be identified by specifying a <code>textField</code> value.</p> <p><b>a wildcard value</b> any valid pathname with a wildcard ("glob") expression that can be expanded by the operating system. This may consist of multiple file types.</p> <p>xml Basic flat XML documents are supported – those of the kind supported by the function <code>xmlToDataFrame</code> function of the <b>XML</b> package.</p> <p>zip zip archive file, containing *.txt files. This may be a URL to a zip file.</p>
<code>textField</code>	a variable (column) name or column number indicating where to find the texts that form the documents for the corpus. This must be specified for file types <code>.csv</code> and <code>.json</code> .
<code>encodingFrom</code>	a single character value specifying the input file encoding, or a vector of character values where each element corresponds to a single file, if a filemask or multiple filenames are supplied as <code>file</code> . These work in the same way as the <code>encoding</code> argument supplied to <code>file</code> , which uses the naming conventions and conversion functions of <code>iconv</code> . If no <code>encodingFrom</code> argument is supplied, then the default encoding is assumed, which may very well be incorrect. Currently, this argument only works when reading text (txt) files.
<code>encodingTo</code>	an optional value that can specify the encoding you wish the files to be converted to, but we strongly encourage you to use the default of UTF-8.
<code>cache</code>	If TRUE, write the object to a temporary file and store the temporary filename in the <code>corpusSource-class</code> object definition. If FALSE, return the data in the object. Caching the file provides a way to read in very large quantities of textual data without storing two copies in memory: one as a <code>corpusSource-class</code> object and the second as a <code>corpus</code> class object. It also provides a way to try different settings of encoding conversion when creating a corpus from a <code>corpusSource-class</code> object, without having to load in all of the source data again.
<code>docvarsfrom</code>	used to specify that docvars should be taken from the filenames, when the <code>textfile</code> inputs are filenames and the elements of the filenames are document variables, separated by a delimiter ( <code>dvsep</code> ). This allows easy assignment of docvars from filenames such as 1789-Washington.txt, 1793-Washington, etc. by <code>dvsep</code> or from meta-data embedded in the text file header (headers).
<code>dvsep</code>	separator used in filenames to delimit docvar elements if <code>docvarsfrom="filenames"</code> is used
<code>docvarnames</code>	character vector of variable names for docvars, if <code>docvarsfrom</code> is specified. If this argument is not used, default docvar names will be used ( <code>docvar1</code> , <code>docvar2</code> , ...).
...	additional arguments passed through to other functions

## Details

The constructor does not store a copy of the texts, but rather reads in the texts and associated data, and saves them to a temporary disk file whose location is specified in the `corpusSource-class` object. This prevents a complete copy of the object from cluttering the global environment and consuming additional space. This does mean however that the state of the file containing the source data will



not be cross-platform and may not be persistent across sessions. So the recommended usage is to load the data into a corpus in the same session in which `textfile` is called.

### Value

an object of class `corpusSource-class` that can be read by `corpus` to construct a corpus

### Author(s)

Kenneth Benoit and Paul Nulty

### Examples

```
## Not run: # Twitter json
mytf1 <- textfile("http://www.kenbenoit.net/files/tweets.json")
summary(corpus(mytf1), 5)
# generic json - needs a textfield specifier
mytf2 <- textfile("http://www.kenbenoit.net/files/sotu.json",
                  textfield = "text")
summary(corpus(mytf2))
# text file
mytf3 <- textfile("https://www.gutenberg.org/cache/epub/2701/pg2701.txt", cache = FALSE)
summary(corpus(mytf3))
# XML data
mytf6 <- textfile("http://www.kenbenoit.net/files/plant_catalog.xml",
                  textfield = "COMMON")
summary(corpus(mytf6))
# csv file
write.csv(data.frame(inaugSpeech = texts(inaugCorpus), docvars(inaugCorpus)),
          file = "/tmp/inaugTexts.csv", row.names = FALSE)
mytf7 <- textfile("/tmp/inaugTexts.csv", textfield = "inaugSpeech")
summary(corpus(mytf7))

## End(Not run)
```

---

textmodel

*fit a text model*


---

### Description

Fit a text model to a dfm. Creates an object of virtual class `textmodel_fitted-class`, whose exact properties (slots and methods) will depend on which model was called (see model types below).

### Usage

```
textmodel(x, y = NULL, data = NULL, model = c("wordscores", "NB",
        "wordfish", "lda", "ca"), ...)

## S4 method for signature 'dfm,ANY,missing,character'
```

```
textmodel(x, y = NULL, data = NULL,
  model = c("wordscores", "NB", "wordfish", "lda", "ca"), ...)

## S4 method for signature 'formula,missing,dfm,character'
textmodel(x, y = NULL,
  data = NULL, model = c("wordscores", "NB", "wordfish", "lda", "ca"), ...)
```

## Arguments

<code>x</code>	a quanteda <a href="#">dfm</a> object containing feature counts by document
<code>y</code>	for supervised models, a vector of class labels or values for training the model, with NA for documents to be excluded from the training set; for unsupervised models, this will be left NULL
<code>data</code>	dfm or data.frame from which to take the formula
<code>model</code>	the model type to be fit. Currently implemented methods are:  wordscores Fits the "wordscores" model of Laver, Benoit, and Garry (2003). Options include the original linear scale of LBG or the logit scale proposed by Beauchamps (2001). See <a href="#">textmodel_wordscores</a> . ca Correspondence analysis scaling of the dfm. NB Fits a Naive Bayes model to the dfm, with options for smoothing, setting class priors, and a choice of multinomial or binomial probabilities. wordfish Fits the "wordfish" model of Slapin and Proksch (2008). lda Fit a topic model based on latent Dirichlet allocation. Temporarily removed. kNN k-nearest neighbour classification, coming soon.
<code>...</code>	additional arguments to be passed to specific model types
<code>formula</code>	An object of class <a href="#">formula</a> of the form $y \sim x_1 + x_2 + \dots$ (Interactions are not currently allowed for any of the models implemented.) The $x$ variable(s) is typically a <a href="#">dfm</a> , and the $y$ variable a vector of class labels or training values associated with each document.

## Value

a textmodel class list, containing the fitted model and additional information specific to the model class. See the methods for specific models, e.g. [textmodel\\_wordscores](#), etc.

## Class hierarchy

Here will go the description of the class hierarchy that governs dispatch for the predict, print, summary methods, since this is not terribly obvious. (Blame it on the S3 system.)

## See Also

[textmodel](#), [textmodel\\_wordscores](#)

**Examples**

```

ieDfm <- dfm(ie2010Corpus, verbose=FALSE)
refscores <- c(rep(NA, 4), -1, 1, rep(NA, 8))
ws <- textmodel(ieDfm, refscores, model="wordscores", smooth=1)

# alternative formula notation - but slower
# need the - 1 to remove the intercept, as this is literal formula notation
wsform <- textmodel(refscores ~ . - 1, data=ieDfm, model="wordscores", smooth=1)
identical(ws@Sw, wsform@Sw) # compare wordscores from the two models

# compare the logit and linear wordscores
bs <- textmodel(ieDfm[5:6,], refscores[5:6], model="wordscores", scale="logit", smooth=1)
plot(ws@Sw, bs@Sw, xlim=c(-1, 1), xlab="Linear word score", ylab="Logit word score")

## Not run: wf <- textmodel(ieDfm, model="wordfish", dir = c(6,5))
wf
## End(Not run)

```

---

textmodel\_ca

*correspondence analysis of a document-feature matrix*


---

**Description**

textmodel\_ca implements correspondence analysis scaling on a [dfm](#). Currently the method is a wrapper to [ca.matrix](#) in the **ca** package.

**Usage**

```
textmodel_ca(data, smooth = 0, ...)
```

**Arguments**

data	the dfm on which the model will be fit
smooth	a smoothing parameter for word counts; defaults to zero.
...	additional arguments passed to <a href="#">ca.matrix</a>

**Author(s)**

Kenneth Benoit

**Examples**

```

ieDfm <- dfm(ie2010Corpus)
wca <- textmodel_ca(ieDfm)
summary(wca)

```

---

textmodel\_fitted-class

*the fitted textmodel classes*


---

## Description

The textmodel virtual class is a parent class for more specific fitted text models, which are the result of a quantitative text analysis applied to a document-feature matrix.

## Details

Available types currently include...

## Slots

dfm a [dfm-class](#) document-feature matrix

---

textmodel\_wordfish

*wordfish text model*


---

## Description

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

## Usage

```
textmodel_wordfish(data, dir = c(1, 2), priors = c(Inf, Inf, 3, 1),
  tol = c(1e-06, 1e-08), dispersion = "poisson")
```

```
## S3 method for class 'textmodel_wordfish_fitted'
print(x, n = 30L, ...)
```

```
## S4 method for signature 'textmodel_wordfish_fitted'
show(object)
```

```
## S4 method for signature 'textmodel_wordfish_predicted'
show(object)
```

**Arguments**

data	the dfm on which the model will be fit
dir	set global identification by specifying the indexes for a pair of documents such that $\hat{\theta}_{dir[1]} < \hat{\theta}_{dir[2]}$ .
priors	priors for $\theta_i$ , $\alpha_i$ , $\psi_j$ , and $\beta_j$ where $i$ indexes documents and $j$ indexes features
tol	tolerances for convergence (explain why a pair)
dispersion	sets whether a quasi-poisson quasi-likelihood should be used based on a single dispersion parameter ("single"), dispersion parameters for each work ("byterm" or "bytermfloor"), or not ("none")
x	for print method, the object to be printed
n	max rows of dfm to print
...	additional arguments passed to other functions
object	wordfish fitted or predicted object to be shown

**Details**

The returns match those of Will Lowe's R implementation of wordfish (see the austin package), except that here we have renamed words to be features. (This return list may change.) We have also followed the practice begun with Slapin and Proksch's early implementation of the model that used a regularization parameter of  $se(\sigma) = 3$ , through the third element in priors.

**Value**

An object of class textmodel\_fitted\_wordfish. This is a list containing:

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

**Author(s)**

Benjamin Lauderdale and Kenneth Benoit

**References**

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

**Examples**

```
ie2010dfm <- dfm(ie2010Corpus, verbose=FALSE)
wfmodel <- textmodel_wordfish(LBGexample, dir = c(6,5))
wfmodel

## Not run: if (require(austin)) {
  wfmodelAustin <- wordfish(quanteda::as.wfm(LBGexample), dir = c(6,5))
  cor(wfmodel@theta, wfmodelAustin$theta)
}
## End(Not run)
```

---

textmodel\_wordscores    *Wordscores text model*

---

**Description**

textmodel\_wordscores implements Laver, Benoit and Garry's (2003) wordscores method for scaling of a single dimension. This can be called directly, but the recommended method is through [textmodel](#).

**Usage**

```
textmodel_wordscores(data, scores, scale = c("linear", "logit"), smooth = 0)

## S3 method for class 'textmodel_wordscores_fitted'
predict(object, newdata = NULL,
  rescaling = "none", level = 0.95, verbose = TRUE, ...)

## S3 method for class 'textmodel_wordscores_fitted'
print(x, n = 30L, digits = 2, ...)

## S4 method for signature 'textmodel_wordscores_fitted'
show(object)

## S4 method for signature 'textmodel_wordscores_predicted'
show(object)

## S3 method for class 'textmodel_wordscores_predicted'
print(x, ...)
```

**Arguments**

data	the dfm on which the model will be fit. Does not need to contain only the training documents, since the index of these will be matched automatically.
scores	vector of training scores associated with each document identified in refData
scale	classic LBG linear posterior weighted word class differences, or logit scale of log posterior differences

<code>smooth</code>	a smoothing parameter for word counts; defaults to zero for the to match the LBG (2003) method.
<code>object</code>	the fitted wordscores textmodel on which prediction will be made
<code>newdata</code>	dfm on which prediction should be made
<code>rescaling</code>	none for "raw" scores; lbg for LBG (2003) rescaling; or mv for the rescaling proposed by Martin and Vanberg (2007). (Note to authors: Provide full details here in documentation.)
<code>level</code>	probability level for confidence interval width
<code>verbose</code>	If TRUE, output status messages
<code>...</code>	additional arguments passed to other functions
<code>x</code>	for print method, the object to be printed
<code>n</code>	max rows of dfm to print
<code>digits</code>	number of decimal places to print for print methods

## Details

Fitting a `textmodel_ordscores` results in an object of class `textmodel_ordscores_fitted` containing the following slots:

## Value

The `predict` method for a wordscores fitted object returns a `data.frame` whose rows are the documents fitted and whose columns contain the scored textvalues, with the number of columns depending on the options called (for instance, how many rescaled scores, and whether standard errors were requested.) (Note: We may very well change this soon so that it is a list similar to other existing fitted objects.)

## Slots

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

## Author(s)

Kenneth Benoit

## References

- Laver, Michael, Kenneth R Benoit, and John Garry. 2003. "Extracting Policy Positions From Political Texts Using Words as Data." *American Political Science Review* 97(02): 311-31
- Beauchamp, N. 2012. "Using Text to Scale Legislatures with Uninformative Voting." New York University Mimeo.
- Martin, L W, and G Vanberg. 2007. "A Robust Transformation Procedure for Interpreting Political Text." *Political Analysis* 16(1): 93-100.
- Laver, Michael, Kenneth R Benoit, and John Garry. 2003. "Extracting Policy Positions From Political Texts Using Words as Data." *American Political Science Review* 97(02): 311-31.
- Martin, L W, and G Vanberg. 2007. "A Robust Transformation Procedure for Interpreting Political Text." *Political Analysis* 16(1): 93-100.

## Examples

```
(ws <- textmodel(LBGexample, c(seq(-1.5, 1.5, .75), NA), model="wordscores"))
predict(ws)
predict(ws, rescaling="mv")
predict(ws, rescaling="lbg")

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

---

texts	<i>get corpus texts</i>
-------	-------------------------

---

## Description

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

## Usage

```
texts(x, groups = NULL, ...)

## S3 method for class 'corpus'
texts(x, groups = NULL, ...)

## S3 method for class 'character'
texts(x, groups = NULL, ...)

## S3 method for class 'corpusSource'
texts(x, groups = NULL, ...)
```



**Arguments**

<code>x</code>	A quanteda corpus object
<code>groups</code>	character vector containing the names of document variables in a corpus, or a factor equal in length to the number of documents, used for aggregating the texts through concatenation. If <code>x</code> is of type character, then <code>groups</code> must be a factor.
<code>...</code>	unused

**Value**

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

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

**Examples**

```
nchar(texts(subset(inaugCorpus, Year < 1806)))

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

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

---

tokenize	<i>tokenize a set of texts</i>
----------	--------------------------------

---

**Description**

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

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

**Usage**

```
tokenize(x, ...)

## S3 method for class 'character'
tokenize(x, what = c("word", "sentence", "character",
  "fastestword", "fasterword"), removeNumbers = FALSE, removePunct = FALSE,
  removeSeparators = TRUE, removeTwitter = FALSE, removeHyphens = FALSE,
  ngrams = 1L, skip = 0L, concatenator = "_", simplify = FALSE,
  verbose = FALSE, ...)

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

is.tokenizedTexts(x)
```

## Arguments

<code>x</code>	The text(s) or corpus to be tokenized
<code>...</code>	additional arguments not used
<code>what</code>	the unit for splitting the text, defaults to "word". Available alternatives are <code>c("character", "word", "line_break", "sentence")</code> . See <a href="#">stringi-search-boundaries</a> .
<code>removeNumbers</code>	remove tokens that consist only of numbers, but not words that start with digits, e.g. 2day
<code>removePunct</code>	remove all punctuation
<code>removeSeparators</code>	remove Separators and separator characters (spaces and variations of spaces, plus tab, newlines, and anything else in the Unicode "separator" category) when <code>removePunct=FALSE</code> . Only applicable for <code>what = "character"</code> (when you probably want it to be FALSE) and for <code>what = "word"</code> (when you probably want it to be TRUE). Note that if <code>what = "word"</code> and you set <code>removePunct = TRUE</code> , then <code>removeSeparators</code> has no effect. Use carefully.
<code>removeTwitter</code>	remove Twitter characters @ and #; set to FALSE if you wish to eliminate these.
<code>removeHyphens</code>	if TRUE, split words that are connected by hyphenation and hyphenation-like characters in between words, e.g. "self-storage" becomes <code>c("self", "storage")</code> . Default is FALSE to preserve such words as is, with the hyphens. Only applies if <code>what = "word"</code> .
<code>ngrams</code>	integer vector of the $n$ for $n$ -grams, defaulting to 1 (unigrams). For bigrams, for instance, use 2; for bigrams and unigrams, use 1:2. You can even include irregular sequences such as 2:3 for bigrams and trigrams only. See <a href="#">ngrams</a> .
<code>skip</code>	integer vector specifying the skips for skip-grams, default is 0 for only immediately neighbouring words. Only applies if <code>ngrams</code> is different from the default of 1. See <a href="#">skipgrams</a> .
<code>concatenator</code>	character to use in concatenating $n$ -grams, default is "_", which is recommended since this is included in the regular expression and Unicode definitions of "word" characters
<code>simplify</code>	if TRUE, return a character vector of tokens rather than a list of length <code>ndoc(texts)</code> , with each element of the list containing a character vector of the tokens corresponding to that text.
<code>verbose</code>	if TRUE, print timing messages to the console; off by default

## Details

The tokenizer is designed to be fast and flexible as well as to handle Unicode correctly. Most of the time, users will construct [dfm](#) objects from texts or a corpus, without calling `tokenize()` as an intermediate step. Since `tokenize()` is most likely to be used by more technical users, we have set its options to default to minimal intervention. This means that punctuation is tokenized as well, and that nothing is removed from the

**Value**

A list of length `ndoc(x)` of the tokens found in each text.

a **tokenizedText** (S3) object, essentially a list of character vectors. If `simplify = TRUE` then return a single character vector.

**Note**

This replaces an older function named `clean()`, removed from **quanteda** in version 0.8.1. "Cleaning" by removing certain parts of texts, such as punctuation or numbers, only works on tokenized texts, although texts of any length can be converted to lower case using `toLower`.

**Author(s)**

Ken Benoit and Paul Nulty

**See Also**

[ngrams](#)

**Examples**

```
# same for character vectors and for lists
tokensFromChar <- tokenize(inaugTexts[1:3])
tokensFromCorp <- tokenize(subset(inaugCorpus, Year<1798))
identical(tokensFromChar, tokensFromCorp)
str(tokensFromChar)
# returned as a list
head(tokenize(inaugTexts[57])[[1]], 10)
# returned as a character vector using simplify=TRUE
head(tokenize(inaugTexts[57], simplify=TRUE), 10)

# removing punctuation marks and lowecasing texts
head(tokenize(toLower(inaugTexts[57]), simplify=TRUE, removePunct=TRUE), 30)
# keeping case and punctuation
head(tokenize(inaugTexts[57], simplify=TRUE), 30)
# keeping versus removing hyphens
tokenize("quanteda data objects are auto-loading.", removePunct = TRUE)
tokenize("quanteda data objects are auto-loading.", removePunct = TRUE, removeHyphens = TRUE)

## MORE COMPARISONS
txt <- "##textanalysis is MY <3 4U @myhandle gr8 #stuff :-)"
tokenize(txt, removePunct=TRUE)
tokenize(txt, removePunct=TRUE, removeTwitter=TRUE)
#tokenize("great website http://textasdata.com", removeURL=FALSE)
#tokenize("great website http://textasdata.com", removeURL=TRUE)

txt <- c(text1="This is $10 in 999 different ways,\n up and down; left and right!",
        text2="@kenbenoit working: on #quanteda 2day\t4ever, http://textasdata.com?page=123.")
tokenize(txt, verbose=TRUE)
tokenize(txt, removeNumbers=TRUE, removePunct=TRUE)
tokenize(txt, removeNumbers=FALSE, removePunct=TRUE)
```

```

tokenize(txt, removeNumbers=TRUE, removePunct=FALSE)
tokenize(txt, removeNumbers=FALSE, removePunct=FALSE)
tokenize(txt, removeNumbers=FALSE, removePunct=FALSE, removeSeparators=FALSE)

# character level
tokenize("Great website: http://textasdata.com?page=123.", what="character")
tokenize("Great website: http://textasdata.com?page=123.", what="character",
         removeSeparators=FALSE)

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

# creating ngrams
txt <- toLower(c(mytext1 = "This is a short test sentence.",
                 mytext2 = "Short.",
                 mytext3 = "Short, shorter, and shortest."))
tokenize(txt, removePunct = TRUE)
removeFeatures(tokenize(txt, removePunct = TRUE), stopwords("english"))
tokenize(txt, removePunct = TRUE, ngrams = 2)
tokenize(txt, removePunct = TRUE, ngrams = 1:2)
tokenize(txt, removePunct = TRUE, ngrams = 2, skip = 1, concatenator = " ")
removeFeatures(tokenize(txt, removePunct = TRUE, ngrams = 1:2), stopwords("english"))

```

---

toLower

---

*Convert texts to lower case*


---

## Description

Convert texts or tokens to lower case

## Usage

```

toLower(x, keepAcronyms = FALSE, ...)

## S3 method for class 'character'
toLower(x, keepAcronyms = FALSE, ...)

## S3 method for class 'NULL'
toLower(x, ...)

## S3 method for class 'tokenizedTexts'
toLower(x, keepAcronyms = FALSE, ...)

## S3 method for class 'corpus'
toLower(x, keepAcronyms = FALSE, ...)

```

**Arguments**

x	texts to be lower-cased
keepAcronyms	if TRUE, do not lowercase any all-uppercase words
...	additional arguments passed to <b>stringi</b> functions, (e.g. <code>stri_trans_tolower</code> ), such as <code>locale</code>

**Value**

Texts tranformed into their lowercased versions. If x is a character vector or a corpus, return a lowercased character vector. If x is a list of tokenized texts, then return a list of lower-cased tokenized texts.

**Examples**

```
test1 <- c(text1 = "England and France are members of NATO and UNESCO",
           text2 = "NASA sent a rocket into space.")
toLower(test1)
toLower(test1, keepAcronyms = TRUE)

test2 <- tokenize(test1, removePunct=TRUE)
toLower(test2)
toLower(test2, keepAcronyms = TRUE)
```

---

topfeatures	<i>list the most frequent features</i>
-------------	--

---

**Description**

List the most frequently occurring features in a [dfm](#)

**Usage**

```
topfeatures(x, ...)
```

```
## S3 method for class 'dfm'
topfeatures(x, n = 10, decreasing = TRUE, ci = 0.95, ...)
```

```
## S3 method for class 'dgCMatrix'
topfeatures(x, n = 10, decreasing = TRUE, ...)
```

**Arguments**

x	the object whose features will be returned
...	additional arguments passed to other methods
n	how many top features should be returned
decreasing	If TRUE, return the n most frequent features, if FALSE, return the n least frequent features
ci	confidence interval from 0-1.0 for use if dfm is resampled

**Value**

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

**Examples**

```
topfeatures(dfm(subset(inaugCorpus, Year>1980), verbose=FALSE))
topfeatures(dfm(subset(inaugCorpus, Year>1980), ignoredFeatures=stopwords("english"),
  verbose=FALSE))
# least frequent features
topfeatures(dfm(subset(inaugCorpus, Year>1980), verbose=FALSE), decreasing=FALSE)
```

---

trim	<i>Trim a dfm using threshold-based or random feature selection</i>
------	---

---

**Description**

Returns a document by feature matrix reduced in size based on document and term frequency, and/or subsampling.

**Usage**

```
trim(x, minCount = 1, minDoc = 1, sparsity = NULL, nsample = NULL,
  verbose = TRUE)

## S4 method for signature 'dfm'
trim(x, minCount = 1, minDoc = 1, sparsity = NULL,
  nsample = NULL, verbose = TRUE)

trimdfm(x, ...)
```

**Arguments**

x	document-feature matrix of <a href="#">dfm-class</a>
minCount	minimum count or fraction of features in across all documents
minDoc	minimum number or fraction of documents in which a feature appears
sparsity	equivalent to 1 - minDoc, included for comparison with tm
nsample	how many features to retain (based on random selection)
verbose	print messages
...	only included to allow legacy trimdfm to pass arguments to trim

**Value**

A [dfm-class](#) object reduced in features (with the same number of documents)

**Note**

Trimming a [dfm-class](#) object is an operation based on the values in the document-feature *matrix*. To select subsets of a dfm based on attributes of the features themselves – such as selecting features matching a regular expression, or removing features matching a stopwords list, use [selectFeatures](#).

**Author(s)**

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

**See Also**

[selectFeatures](#)

**Examples**

```
(myDfm <- dfm(inaugCorpus, verbose = FALSE))
# only words occurring >=10 times and in >=2 docs
trim(myDfm, minCount = 10, minDoc = 2)
# only words occurring >=10 times and in at least 0.4 of the documents
trim(myDfm, minCount = 10, minDoc = 0.4)
# only words occurring at least 0.01 times and in >=2 documents
trim(myDfm, minCount = .01, minDoc = 2)
# only words occurring 5 times in 1000
trim(myDfm, minDoc = 0.2, minCount = 0.005)
# sample 50 words occurring at least 20 times each
(myDfmSampled <- trim(myDfm, minCount = 20, nsample = 50))
topfeatures(myDfmSampled)
## Not run:
if (require(tm)) {
  (tmdtm <- convert(myDfm, "tm"))
  removeSparseTerms(tmdtm, 0.7)
  trim(td, minDoc = 0.3)
  trim(td, sparsity = 0.7)
}

## End(Not run)
```

---

ukimmigTexts

*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.

**Format**

A named character vector of plain ASCII texts

## Examples

```
ukimmigCorpus <- corpus(ukimmigTexts, docvars=data.frame(party=names(ukimmigTexts)))
metadoc(ukimmigCorpus, "language") <- "english"
summary(ukimmigCorpus, showmeta = TRUE)
```

---

weight

*Weight the feature frequencies in a dfm by various methods*


---

## Description

Returns a document by feature matrix with the feature frequencies weighted according to one of several common methods.

## Usage

```
weight(x, ...)

## S4 method for signature 'dfm'
weight(x, type = c("frequency", "relFreq", "relMaxFreq",
  "logFreq", "tfidf"), smoothing = 0, normalize = TRUE, verbose = TRUE,
  ...)

tf(x)

tfidf(x)

smoother(x, smoothing)

weighting(object)

## S4 method for signature 'dfm'
weighting(object)
```

## Arguments

x	document-feature matrix created by <a href="#">dfm</a>
...	not currently used
type	The weighting function to apply to the dfm. One of: <ul style="list-style-type: none"> <li>• normTf - Length normalization: dividing the frequency of the feature by the length of the document)</li> <li>• logTf - The natural log of the term frequency</li> <li>• tf-idf - Term-frequency * inverse document frequency. For a full explanation, see, for example, <a href="http://nlp.stanford.edu/IR-book/html/htmledition/term-frequency-and-weighting-1.html">http://nlp.stanford.edu/IR-book/html/htmledition/term-frequency-and-weighting-1.html</a>. This implementation will not return negative values.</li> </ul>



	<ul style="list-style-type: none"> <li>• maxTf - The term frequency divided by the frequency of the most frequent term in the document</li> <li>• ppmi - Positive Pointwise Mutual Information</li> </ul>
smoothing	amount to apply as additive smoothing to the document-feature matrix prior to weighting, default is 0.5, set to smoothing=0 for no smoothing.
normalize	if TRUE (default) then normalize the dfm by relative term frequency prior to computing tfidf
verbose	if TRUE output status messages
object	the dfm object for accessing the weighting setting

### Details

tf is a shortcut for `weight(x, "relFreq")`

tfidf is a shortcut for `weight(x, "tfidf")`

`smoother(x, smoothing)` is a shortcut for `weight(x, "frequency", smoothing)`

weighting queries (but cannot set) the weighting applied to the dfm.

### Value

The dfm with weighted values

weighting returns a character object describing the type of weighting applied to the dfm.

### Author(s)

Paul Nulty and Kenneth Benoit

### References

Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze. Introduction to information retrieval. Vol. 1. Cambridge: Cambridge university press, 2008.

### Examples

```
dtm <- dfm(inaugCorpus)
x <- apply(dtm, 1, function(tf) tf/max(tf))
topfeatures(dtm)
normDtm <- weight(dtm)
topfeatures(normDtm)
maxTfDtm <- weight(dtm, type="relMaxFreq")
topfeatures(maxTfDtm)
logTfDtm <- weight(dtm, type="logFreq")
topfeatures(logTfDtm)
tfidfDtm <- weight(dtm, type="tfidf")
topfeatures(tfidfDtm)

# combine these methods for more complex weightings, e.g. as in Section 6.4 of
# Introduction to Information Retrieval
logTfDtm <- weight(dtm, type="logFreq")
```

```
wfidfDtm <- weight(logTfDtm, type="tfidf", normalize=FALSE)
```

wordlists

*word lists used in some readability indexes*

### Description

`wordlists` is a named list of character vectors where each list element corresponds to a different readability index. These are:

`DaleChall` The long Dale-Chall list of 3,000 familiar (English) words needed to compute the Dale-Chall Readability Formula.

`Spache` The revised Spache word list (see Klare 1975, 73) needed to compute the Spache Revised Formula of readability (Spache 1974).

### References

Chall, J. S., & Dale, E. 1995. *Readability Revisited: The New Dale-Chall Readability Formula*. Brookline Books.

Klare, G. R. 1975. "Assessing readability." *Reading Research Quarterly* 10(1): 62–102.

Spache, G. 1953. "A new readability formula for primary-grade reading materials." *The Elementary School Journal* 53: 410–413.

wordstem

*stem words*

### 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

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

## S3 method for class 'character'
wordstem(x, language = "porter")

## S3 method for class 'tokenizedTexts'
wordstem(x, language = "porter")

## S3 method for class 'dfm'
wordstem(x, language = "porter")
```

**Arguments**

x	a character vector or corpus, whose word stems are to be removed
language	the name of a recognized language, as returned by <a href="#">getStemLanguages</a> , or a two- or three-letter ISO-639 code corresponding to one of these languages (see references for the list of codes)

**Value**

A character vector with as many elements as there are in the input vector with the corresponding elements being the stem of the word. Elements of the vector are converted to UTF-8 encoding before the stemming is performed, and the returned elements are marked as such when they contain non-ASCII characters.

**References**

<http://snowball.tartarus.org/>

[http://www.iso.org/iso/home/standards/language\\_codes.htm](http://www.iso.org/iso/home/standards/language_codes.htm) for the ISO-639 language codes

**See Also**

[wordStem](#)

**Examples**

```
#' Simple example
wordstem(c("win", "winning", "wins", "won", "winner"))
```

# Index

`+`, `dfmDense`, numeric-method (dfm-class), 18  
`+`, `dfmSparse`, numeric-method (dfm-class), 18  
`+`, numeric, `dfmDense`-method (dfm-class), 18  
`+`, numeric, `dfmSparse`-method (dfm-class), 18  
`+.corpus` (corpus), 11  
`.stopwords` (stopwords), 59  
`[,dfmDense`, index, index, logical-method (dfm-class), 18  
`[,dfmDense`, index, index, missing-method (dfm-class), 18  
`[,dfmDense`, index, missing, logical-method (dfm-class), 18  
`[,dfmDense`, index, missing, missing-method (dfm-class), 18  
`[,dfmDense`, missing, index, logical-method (dfm-class), 18  
`[,dfmDense`, missing, index, missing-method (dfm-class), 18  
`[,dfmDense`, missing, missing, logical-method (dfm-class), 18  
`[,dfmDense`, missing, missing, missing-method (dfm-class), 18  
`[,dfmSparse`, index, index, logical-method (dfm-class), 18  
`[,dfmSparse`, index, index, missing-method (dfm-class), 18  
`[,dfmSparse`, index, missing, logical-method (dfm-class), 18  
`[,dfmSparse`, index, missing, missing-method (dfm-class), 18  
`[,dfmSparse`, missing, index, logical-method (dfm-class), 18  
`[,dfmSparse`, missing, index, missing-method (dfm-class), 18  
`[,dfmSparse`, missing, missing, logical-method (dfm-class), 18  
`[,dfmSparse`, missing, missing, missing-method (dfm-class), 18  
`[.corpus` (corpus), 11  
`[.dfm` (dfm-class), 18  
`applyDictionary`, 4  
`as.data.frame`, dfm-method (dfm-class), 18  
`as.dfm` (dfm), 15  
`as.DocumentTermMatrix` (convert), 9  
`as.matrix`, dfm-method (dfm-class), 18  
`as.wfm` (convert), 9  
`ca.matrix`, 67  
`cbind`, 5  
`cbind.dfm`, 5  
`cbind2`, 5  
`changeunits`, 6, 53  
`clean` (tokenize), 73  
`collocations`, 7, 43, 48  
`colSums`, `dfmDense`-method (dfm-class), 18  
`colSums`, `dfmSparse`-method (dfm-class), 18  
`comparison.cloud`, 44  
`convert`, 9  
`corpus`, 11, 11, 26, 47, 64, 65  
`corpusSource-class`, 11, 12, 14, 26, 64, 65  
`data.frame`, 13  
`describeTexts` (summary.corpus), 60  
`dfm`, 5, 10, 13, 15, 17, 21, 23, 24, 28, 32, 42, 43, 50, 54–58, 66, 67, 74, 77, 80  
`dfm-class`, 9, 15, 17, 18, 23, 24, 28, 29, 54, 68, 78, 79  
`dfm2ldaformat` (convert), 9  
`dfmDense-class` (dfm-class), 18  
`dfmSparse-class` (dfm-class), 18  
`dgCMatrix-class`, 21  
`dgeMatrix-class`, 21  
`dictionary`, 4, 22, 43  
`docfreq`, 23

- docfreq,dfm,missing-method (docfreq), 23
- docfreq,dfm,numeric-method (docfreq), 23
- docfreq,dfmDense,missing-method (docfreq), 23
- docfreq,dfmDense,numeric-method (docfreq), 23
- docfreq,dfmSparse,missing-method (docfreq), 23
- docfreq,dfmSparse,numeric-method (docfreq), 23
- docnames, 24
- docnames<- (docnames), 24
- document-feature matrix object, 33
- DocumentTermMatrix, 10
- docvars, 13, 14, 25
- docvars<- (docvars), 25
- drop, 12
- encodedTexts, 26
- encoding, 26
- englishSyllables (syllables), 61
- exampleString, 27
- features, 28
- file, 22, 64
- formula, 66
- getStemLanguages, 83
- head.dfm, 28
- iconv, 22, 64
- ie2010Corpus, 29
- iebudgets (ie2010Corpus), 29
- inaugCorpus, 30
- inaugTexts (inaugCorpus), 30
- is.corpus (corpus), 11
- is.dfm (dfm), 15
- is.tokenizedTexts (tokenize), 73
- kwic, 30, 45
- LBGexample, 32
- lda.collapsed.gibbs.sampler, 10
- lexdiv, 32
- Matrix, 16
- Matrix-class, 18, 21
- metacorporus, 14, 35
- metacorporus<- (metacorporus), 35
- metadoc, 13, 14, 36
- metadoc<- (metadoc), 36
- ndoc, 37, 74, 75
- nfeature (ndoc), 37
- ngrams, 8, 38, 39, 74, 75
- nsentence, 40
- ntoken, 37, 41
- ntype (ntoken), 41
- phrasetotoken, 42
- phrasetotoken,character,collocations-method (phrasetotoken), 42
- phrasetotoken,character,dictionary-method (phrasetotoken), 42
- phrasetotoken.corpus (phrasetotoken), 42
- plot, 45
- plot.dfm, 43
- plot.kwic, 44
- pr\_DB, 56, 57
- predict.textmodel\_wordscores\_fitted (textmodel\_wordscores), 70
- print,dfm-method (print.dfm), 45
- print,dfmDense-method (print.dfm), 45
- print,dfmSparse-method (print.dfm), 45
- print.dfm, 45
- print.kwic (kwic), 30
- print.settings (settings), 55
- print.textmodel\_wordfish\_fitted (textmodel\_wordfish), 68
- print.textmodel\_wordscores\_fitted (textmodel\_wordscores), 70
- print.textmodel\_wordscores\_predicted (textmodel\_wordscores), 70
- print.tokenizedTexts, 46
- quanteda-package, 3, 30
- quantedaformat2dtm (convert), 9
- readability, 46
- regex, 52
- removeFeatures, 48, 54
- rowSums,dfmDense-method (dfm-class), 18
- rowSums,dfmSparse-method (dfm-class), 18
- sample, 50
- sample (sample.corpus), 49
- sample.corpus, 49
- scrabble, 51

