Package 'quanteda'

October 31, 2016

Type Package

Title Quantitative Analysis of Textual Data

Author Kenneth Benoit [aut, cre],
Paul Nulty [aut],
Kohei Watanabe [ctb],
Benjamin Lauderdale [ctb],
Adam Obeng [ctb],
Pablo Barberá [ctb],
Will Lowe [ctb]

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

Version 0.9.8.5

Date 2016-10-28

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

License GPL-3

Depends R (>= 3.2.2)

Imports methods, utils, stats, Matrix (>= 1.2), data.table (>= 1.9.6), SnowballC, wordcloud, proxy, parallel, Rcpp, ca, stringi, httr

LinkingTo Rcpp, RcppArmadillo

Suggests knitr, rmarkdown, lda, topicmodels, jsonlite (>= 0.9.10), streamR, tm (>= 0.6), slam, XML, testthat, RColorBrewer, ggplot2 (>= 2.1.0), grid

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' 'corpus.R' 'collocations.R' 'converters.R' 'dataDocs.R' 'describe-texts.R' 'dfm-classes.R' 'dfm-main.R' 'dfm-methods.R' 'dfm-weighting.R' 'encoding.R'

'findSequences.R' 'joinTokens.R' 'kwic.R' 'lexdiv.R' 'ngrams.R'
'phrases.R' 'plots.R' 'quanteda.R' 'readability.R' 'resample.R'
'selectFeatures-old.R' 'selectFeatures.R' 'settings.R'
'similarity.R' 'stopwords.R' 'syllables.R' 'textfile.R'
'textmodel-NB.R' 'textmodel-ca.R' 'textmodel-generics.R'
'textmodel-wordfish.R' 'textmodel-wordscores.R' 'toLower.R'
'tokenize.R' 'tokenize_outtakes.R' 'util.R' 'wordstem.R'
'zzz.R'

RoxygenNote 5.0.1

SystemRequirements C++11

NeedsCompilation yes

Repository CRAN

Date/Publication 2016-10-31 23:44:48

R topics documented:

quanteda-package 4
applyDictionary 5
as.data.frame,dfm-method 6
cbind.dfm 7
changeunits 8
collocations 9
compress 11
convert 12
corpus 14
corpusSource-class 18
dfm 19
dfm-class 22
dictionary 26
docfreq 27
docnames 29
docvars 30
encodedTextFiles 31
encodedTexts 32
encoding 32
exampleString 33
features 34
findSequences 34
head.dfm 35
head.tokenSequences 36
ie2010Corpus 37
inaugCorpus 37
joinTokens 38
kwic 38
LBGexample 40
lexdiv 40

metacorpus . . . 43
metadoc . . . 44
mobydickText . . . 45
ndoc . . . 45
ngrams . . . 46
nsentence . . . 48
ntoken . . . 49
phrasetotoken . . . 50
plot.dfm . . . 52
plot.kwic . . . 53
predict.textmodel_NB_fitted . . . 54
print.dfm . . . 55
print.tokenizedTexts . . . 56
print.tokenSequences . . . 56
readability . . . 57
removeFeatures . . . 58
sample . . . 59
scrabble . . . 60
segment . . . 61
selectFeatures . . . 63
selectFeaturesOLD . . . 66
settings . . . 67
show,dictionary-method . . . 68
similarity . . . 69
sort.dfm . . . 70
stopwords . . . 71
subset.corpus . . . 72
summary.corpus . . . 73
syllables . . . 74
tail.tokenSequences . . . 75
textfile . . . 76
textmodel . . . 79
textmodel_ca . . . 80
textmodel_fitted-class . . . 81
textmodel_NB . . . 81
textmodel_wordfish . . . 83
textmodel_wordscores . . . 85
texts . . . 87
tf . . . 89
tfidf . . . 90
tokenize . . . 91
toLower . . . 95
topfeatures . . . 96
trim . . . 97
ukimmigTexts . . . 99
weight . . . 99
wordlists . . . 101
wordstem . . . 101

Index **103**

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

Description

An R package for the quantitative analysis of textual data

Author(s)

Ken Benoit and Paul Nulty

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

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

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

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

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

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

Additional features of **quanteda** include:

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

`applyDictionary` *apply a dictionary or thesaurus to an object*

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 dictionary-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 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 "dictionary" operates more like a thesarus. If a thesaurus-like application is desired, set `exclusive = 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)
```

---

as.data.frame,dfm-method

coerce a dfm to a data.frame

Description

Method for coercing a dfm-class object to a data.frame

Usage

```
## S4 method for signature 'dfm'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

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

optional	not applicable to this method
...	not used for this method

Examples

```
inaugDfm <- dfm(inaugTexts[1:5], verbose = FALSE)
as.data.frame(inaugDfm[, 1:10])
str(as.data.frame(inaugDfm))
as.data.frame(inaugDfm[, 1:10], row.names = FALSE)
```

---

cbind.dfm *Combine dfm objects by Rows or Columns*

Description

Take a sequence of dfm-class objects and combine by columns or rows, returning a dfm with the combined documents or features, respectively.

Usage

```
## S3 method for class 'dfm'
cbind(...)

## S3 method for class 'dfm'
rbind(...)
```

Arguments

...	dfm objects to be joined column-wise (cbind) or row-wise (rbind) to the first

Details

`cbind(x, y, ...)` combines dfm objects by columns, returning a dfm object with combined features from input dfm objects. Note that this should be used with extreme caution, as joining dfms with different documents will result in a new row with the docname(s) of the first dfm, merging in those from the second. Furthermore, if features are shared between the dfms being cbinded, then duplicate feature labels will result. In both instances, warning messages will result.

`rbind(x, y, ...)` combines dfm objects by rows, returning a dfm object with combined features from input dfm objects. Features are matched between the two dfm objects, so that the order and names of the features do not need to match. The order of the features in the resulting dfm is not guaranteed. The attributes and settings of this new dfm are not currently preserved.

Examples

```
# cbind() for dfm objects
(dfm1 <- dfm("This is one sample text sample.", verbose = FALSE))
(dfm2 <- dfm("More words here.", verbose = FALSE))
cbind(dfm1, dfm2)

# rbind() for dfm objects
(dfm1 <- dfm(c(doc1 = "This is one sample text sample."), verbose = FALSE))
(dfm2 <- dfm(c(doc2 = "One two three text text."), verbose = FALSE))
(dfm3 <- dfm(c(doc3 = "This is the fourth sample text."), verbose = FALSE))
rbind(dfm1, dfm2)
rbind(dfm1, dfm2, dfm3)
```

---

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. (Because the corpus object records its current "units" status, there is no `from` option, only `to`.)

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.",
                     textwo = "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 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 by default (`spanPunct = FALSE`) 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, toLower = TRUE,
  punctuation = c("dontspan", "ignore", "include"), ...)

## S3 method for class 'character'
collocations(x, method = c("lr", "chi2", "pmi", "dice",
  "all"), size = 2, n = NULL, toLower = TRUE,
  punctuation = c("dontspan", "ignore", "include"), ...)

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

Arguments

x	a text, a character vector of texts, or a corpus
...	additional parameters passed to tokenize
method	association measure for detecting collocations. Let i index documents, and j index features, n_{ij} refers to observed counts, and m_{ij} the expected counts in a collocations frequency table of dimensions $(J - size + 1)^2$. Available measures are computed as:
	"lr" The likelihood ratio statistic G^2, computed as: $$2 * \sum_i \sum_j (n_{ij} * log \frac{n_{ij}}{m_{ij}})$$
	"chi2" Pearson's χ^2 statistic, computed as: $$\sum_i \sum_j \frac{(n_{ij} - m_{ij})^2}{m_{ij}}$$
	"pmi" point-wise mutual information score, computed as log n_{11}/m_{11}
	"dice" the Dice coefficient, computed as $n_{11}/n_{1\cdot} + n_{\cdot 1}$
	"all" returns all of the above
size	length of the collocation. Only bigram (n=2) and trigram (n=3) collocations are currently implemented. 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.
toLower	convert collocations to lower case if TRUE (default)
punctuation	how to handle tokens separated by punctuation characters. Options are:
	dontspan do not form collocations from tokens separated by punctuation characters (default)
	ignore ignore punctuation characters when forming collocations, meaning that collocations will include those separated by punctuation characters in the text. The punctuation characters themselves are not returned.
	include do not treat punctuation characters specially, meaning that collocations will include punctuation characters as tokens

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, punctuation = "dontspan") # default
collocations(txt, punctuation = "dontspan", removePunct = TRUE)  # includes "testing looking"
collocations(txt, punctuation = "ignore", removePunct = TRUE)    # same as previous
collocations(txt, punctuation = "include", removePunct = FALSE)  # keep punctuation as tokens

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)
```

---

compress *compress a dfm by combining similarly named dimensions*

Description

"Compresses" a dfm whose dimension names are the same, for either documents or features. This may happen, for instance, if features are made equivalent through application of a thesaurus. It may also occur after lower-casing or stemming the features of a dfm, but this should only be done in very rare cases (approaching never: it's better to do this *before* constructing the dfm.) It could also be needed , after a `cbind.dfm` or `rbind.dfm` operation.

Usage

```
compress(x, ...)

## S3 method for class 'dfm'
compress(x, margin = c("both", "documents", "features"), ...)
```

Arguments

x	input object, a dfm
...	additional arguments passed from generic to specific methods
margin	character indicating which margin to compress on, either "documents", "features", or "both" (default)

Examples

```
mat <- rbind(dfm(c("b A A", "C C a b B"), toLower = FALSE, verbose = FALSE),
             dfm("A C C C C C", toLower = FALSE, verbose = FALSE))
colnames(mat) <- toLower(features(mat))
mat
compress(mat, margin = "documents")
compress(mat, margin = "features")
compress(mat)

# no effect if no compression needed
compress(dfm(inaugTexts, verbose = FALSE))
```

---

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"),
  docvars = NULL, ...)

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 lda.collapsed.gibbs.sampler from the **lda** package "tm" a DocumentTermMatrix from the **tm** package "stm" the format for the **stm** package "austin" the wfm format from the **austin** package "topicmodels" the "dtm" format as used by the **topicmodels** package
...	not used here
docvars	optional data.frame of document variables used as the meta information in conversion to the STM package format. This aids in selecting the document variables only corresponding to the documents with non-zero counts.

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

# topicmodels package format
topicmodelsdfm <- quantedaformat2dtm(quantdfm)
identical(topicmodelsdfm, convert(quantdfm, to = "topicmodels"))

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

# 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 quanteda 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, docnames = NULL, docvars = NULL,
  source = NULL, notes = NULL, citation = NULL, ...)

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

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

## S3 method for class 'data.frame'
corpus(x, textField, ...)

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

is.corpus(x)

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

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

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

## S3 method for class 'corpus'
x[[i, ...]]

## S3 replacement method for class 'corpus'
x[[i]] <- value
```

Arguments

x	a source of texts to form the documents in the corpus, a character vector or a corpusSource-class object created using `textfile`.

...	additional arguments
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.
textField	the character name or integer index of the source data.frame indicating the column to be read in as text. This must be of mode character.
c1	corpus one to be added
c2	corpus two to be added
recursive	logical used by 'c()' method, always set to 'FALSE'
i	index for documents or rows of document variables
j	index for column of document variables
drop	if TRUE, return a vector if extracting a single document variable; if FALSE, return it as a single-column data.frame. See drop for further details.
value	a vector that will form a new docvar

Details

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

Indexing a corpus using two indexes (integers or column names) will return the document variables, equivalent to `docvars(x)`. Because a corpus is also a list, it is also possible to access, create, or replace docvars using list notation, e.g. `myCorpus[["newSerialDocvar"]] <- paste0("tag", 1:ndoc(myCorpus))`.

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.

The 'c()' operator is also defined for corpus class objects, and provides an easy way to combine multiple corpus objects.

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 metadata, corpus-level metadata, and default settings for subsequent processing of the corpus. A corpus

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

`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.

When `x` is a `data.frame`, then there is no encoding conversion performed on the character input. It is highly recommended that you detect and convert this input to UTF-8 prior to using it as input for a corpus.

Author(s)

Kenneth Benoit and Paul Nulty

See Also

docvars, metadoc, metacorpus, settings, texts, ndoc, docnames

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)))

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)
}

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

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

# concatenate corpus objects
corpus1 <- corpus(inaugTexts[1:2])
corpus2 <- corpus(inaugTexts[3:4])
corpus3 <- subset(inaugCorpus, President == "Obama")
summary(c(corpus1, corpus2, corpus3))

# ways to index corpus elements
inaugCorpus["1793-Washington"]    # 2nd Washington inaugural speech
inaugCorpus[2]                    # same
ie2010Corpus[, "year"]            # access the docvars from ie2010Corpus
ie2010Corpus[["year"]]            # same

# create a new document variable
ie2010Corpus[["govtopp"]] <- ifelse(ie2010Corpus[["party"]] %in% c("FF", "Greens"),
                                    "Government", "Opposition")
docvars(ie2010Corpus)
```

---

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 *create a document-feature matrix*

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, language = "english", thesaurus = NULL,
  dictionary = NULL, valuetype = c("glob", "regex", "fixed"), ...)

## S3 method for class 'tokenizedTexts'
dfm(x, verbose = TRUE, toLower = FALSE,
  stem = FALSE, ignoredFeatures = NULL, keptFeatures = NULL,
  language = "english", thesaurus = NULL, dictionary = NULL,
  valuetype = c("glob", "regex", "fixed"), ...)

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

is.dfm(x)

as.dfm(x)
```

Arguments

x	corpus or character vector from which to generate the document-feature matrix
...	additional arguments passed to tokenize, which can include for instance ngrams and concatenator for tokenizing multi-token sequences
verbose	display messages if TRUE
toLower	convert texts to lowercase
removeNumbers	remove numbers, see tokenize
removePunct	remove punctuation, see tokenize
removeSeparators	remove separators (whitespace), see tokenize
removeTwitter	if FALSE, preserve # and @ characters, see tokenize
stem	if TRUE, stem words
ignoredFeatures	a character vector of user-supplied features to ignore, such as "stop words". To access one possible list (from any list you wish), use stopwords(). The pattern matching type will be set by valuetype. For behaviour of ingoredFeatures with ngrams > 1, see Details.
keptFeatures	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 keptFeatures = "@*" and make sure that removeTwitter = FALSE as an additional argument passed to tokenize. Note: keptFeatures = "^@\\w+\\b" would be the regular expression version of this matching pattern. The pattern matching type will be set by valuetype.
language	Language for stemming. Choices are danish, dutch, english, finnish, french, german, hungarian, italian, norwegian, porter, portuguese, romanian, russian, spanish, swedish, turkish.
thesaurus	A list of character vector "thesaurus" entries, in a dictionary list format, which operates as a dictionary but without excluding values not matched from the dictionary. Thesaurus keys are converted to upper case to create a feature label in the dfm, as a reminder that this was not a type found in the text, but rather the label of a thesaurus key. For more fine-grained control over this and other aspects of converting features into dictionary/thesaurus keys from pattern matches to values, you can use applyDictionary after creating the dfm.
dictionary	A list of character vector dictionary entries, including regular expressions (see examples)
valuetype	fixed for words as is; "regex" for regular expressions; or "glob" for "glob"-style wildcard. Glob format is the default. See selectFeatures.
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", "notincorpus"),
               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, verbose = FALSE))

# 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 'dfmDense'
t(x)

## S4 method for signature 'dfmSparse'
```

```
t(x)

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

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

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

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

## 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,logical,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,logical,missing,missing'
x[i, j, ..., 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)
```

Arguments

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

```
# coercion to matrix
dfmSparse <- dfm(inaugTexts, verbose = FALSE)
str(as.matrix(dfmSparse))
```

`dictionary` *create a dictionary*

Description

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

Usage

```
dictionary(x = NULL, file = NULL, format = NULL, concatenator = " ",
  toLower = TRUE, encoding = "")
```

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. If not supplied, the format is guessed from the dictionary file's extension. Available options are: `"wordstat"` format used by Provalis Research's Wordstat software `"LIWC"` format used by the Linguistic Inquiry and Word Count software `"yoshikoder"` format used by Yoshikoder software `"lexicoder"` format used by Lexicoder
`concatenator`	the character in between multi-word dictionary values. This defaults to `"_"` except LIWC-formatted files, which defaults to a single space `" "`.
`toLower`	if `TRUE`, convert all dictionary values to lowercase
`encoding`	additional optional encoding value for reading in imported dictionaries. This uses the iconv labels for encoding. See the "Encoding" section of the help for file.

Value

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

References

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

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

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

See Also

dfm

Examples

```
mycorpus <- subset(inaugCorpus, Year>1900)
mydict <- dictionary(list(christmas = c("Christmas", "Santa", "holiday"),
                          opposition = c("Opposition", "reject", "notincorpus"),
                          taxing = "taxing",
                          taxation = "taxation",
                          taxregex = "tax*",
                          country = "united states"))
head(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")
head(dfm(inaugTexts, dictionary=lgdict))

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

---

docfreq *compute the (weighted) document frequency of a feature*

Description

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

Usage

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

## S4 method for signature 'dfm'
docfreq(x, scheme = c("count", "inverse", "inversemax",
  "inverseprob", "unary"), smoothing = 0, k = 0, base = 10,
  threshold = 0, USE.NAMES = TRUE)
```

Arguments

x	a dfm-class document-feature matrix
scheme	type of document frequency weighting
smoothing	added to the quotient before taking the logarithm
k	added to the denominator in the "inverse" weighting types, to prevent a zero document count for a term
base	the base with respect to which logarithms in the inverse document frequency weightings are computed; default is 10 (see Manning, Raghavan, and Schutze 2008, p123).
threshold	numeric value of the threshold *above which* a feature will considered in the computation of document frequency. The default is 0, meaning that a feature's document frequency will be the number of documents in which it occurs greater than zero times.
USE.NAMES	logical; if TRUE attach feature labels as names of the resulting numeric vector
...	not used

Value

a numeric vector of document frequencies for each feature

References

Manning, C. D., Raghavan, P., & Schutze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Examples

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

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

docnames *get or set document names*

Description

Get or set the document names from a corpus or a document-feature matrix. of the dfm object.

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

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 *get or set for document-level variables*

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

## S3 replacement method for class 'corpus'
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

Note

Another way to access and set docvars is through indexing of the corpus j element, such as ie2010Corpus[, c("foren", "name"] or for a single docvar, ie2010Corpus[["name"]]. The latter also permits assignment, including the easy creation of new document varibles, e.g. ie2010Corpus[["newvar"]] <- 1
See [.corpus for details.

Examples

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

# alternative using indexing
```

```
head(inaugCorpus[, "Year"])
inaugCorpus[["President2"]] <- paste("prezTwo", 1:ndoc(inaugCorpus), sep="")
head(docvars(inaugCorpus))
```

---

encodedTextFiles *a .zip file of texts containing a variety of differently encoded texts*

Description

A set of translations of the Universal Declaration of Human Rights, plus one or two other miscellaneous texts, for testing the text input functions that need to translate different input encodings.

Source

The Universal Declaration of Human Rights resources, http://www.ohchr.org/EN/UDHR/Pages/SearchByLang.aspx

Examples

```
## Not run: # unzip the files to a temporary directory
FILEDIR <- tempdir()
unzip(system.file("extdata", "encodedTextFiles.zip", package = "quanteda"), exdir = FILEDIR)

# get encoding from filename
filenames <- list.files(FILEDIR, "\\.txt$")
# strip the extension
filenames <- gsub(".txt$", "", filenames)
parts <- strsplit(filenames, "_")
fileencodings <- sapply(parts, "[", 3)
fileencodings

# find out which conversions are unavailable (through iconv())
cat("Encoding conversions not available for this platform:")
notAvailableIndex <- which(!(fileencodings %in% iconvlist()))
fileencodings[notAvailableIndex]

# try textfile
require(quanteda)
tfile <- textfile(paste0(FILEDIR, "/", "*.txt"))
substring(texts(tfile)[1], 1, 80) # gibberish
substring(texts(tfile)[4], 1, 80) # hex
substring(texts(tfile)[40], 1, 80) # hex

# read them in again
tfile <- textfile(paste0(FILEDIR,  "/", "*.txt"), encoding = fileencodings)
substring(texts(tfile)[1], 1, 80)  # English
substring(texts(tfile)[4], 1, 80)  # Arabic, looking good
substring(texts(tfile)[40], 1, 80) # Cyrillic, looking good
substring(texts(tfile)[7], 1, 80)  # Chinese, looking good
substring(texts(tfile)[26], 1, 80) # Hindi, looking good
```

```
tfile <- textfile(paste0(FILEDIR, "/", "*.txt"), encoding = fileencodings,
                  docvarsfrom = "filenames",
                  docvarnames = c("document", "language", "inputEncoding"))
encodingCorpus <- corpus(tfile, source = "Created by encoding-tests.R")
summary(encodingCorpus)

## End(Not run)
```

---

encodedTexts *encoded texts for testing*

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 *detect the encoding of texts*

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 (re)specified when inputting a set of texts using textfile, prior to constructing a 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 stri_enc_detect

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 *extract the feature labels from a dfm*

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

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

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

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

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

findSequences *find sequences of tokens*

Description

This function automatically identify sequences of tokens. This algorithm is based on Blaheta and Johnson's "Unsupervised Learning of Multi-Word Verbs".

Usage

```
findSequences(x, tokens, count_min, smooth = 0.001, nested = TRUE)
```

Arguments

x	tokenizedTexts objects
tokens	types of token in sequuences
count_min	minimum frequency of sequences
smooth	smoothing factor
nested	collect nested sub-sequence

Examples

```
sents <- tokenize(inaugCorpus, what = "sentence", simplify = TRUE)
tokens <- tokenize(sents, removePunct = TRUE)
tokens <- selectFeatures(tokens, stopwords(), 'remove', padding=TRUE)
types <- unique(unlist(tokens))

# Extracting multi-part nouns
types_upper <- types[stringi::stri_detect_regex(types, "^([A-Z][a-z\\-]{2,})")]
seqs <- findSequences(tokens, types_upper, count_min=2)
head(seqs, 30)

# Types can be any words
types_lower <- types[stringi::stri_detect_regex(types, "^([a-z]+)$") & !types %in%stopwords()]
seqs2 <- findSequences(tokens, types_lower, count_min=3)
head(seqs2, 20)
```

---

head.dfm *Return the first or last part of a dfm*

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, ...)

## S4 method for signature 'dfm'
head(x, n = 6L, nfeature = 6L, ...)

## S4 method for signature 'dfm'
tail(x, n = 6L, nfeature = 6L, ...)

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

Arguments

x	a dfm object
n	a single integer. If positive, size for the resulting object: number of first/last documents for the dfm. If negative, all but the n last/first number of documents of x.
nfeature	the number of features to return, where the resulting object will contain the first ncol features
...	additional arguments passed to other functions

Value

A dfm-class 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)
```

---

head.tokenSequences *print a tokenSequences objects*

Description

print method for a tokenSequences object

Usage

```
## S3 method for class 'tokenSequences'
head(x, ...)
```

Arguments

x	a tokenSequences object created by findSequences
...	further arguments passed to base print method

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 inaugaration 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)
```

joinTokens *join tokens function*

Description

Needs some more explanation

Usage

```
joinTokens(x, sequences, concatenator = "-", valuetype = "fixed",
  verbose = FALSE)
```

Arguments

x	some object
sequences	list of vector of features to concatenate
concatenator	character used for joining tokens
valuetype	how to interpret sequences: "fixed" for words as is; "regex" for regular expressions; or "glob" for "glob"-style wildcard
verbose	display progress

Examples

```
toks <- tokenize(inaugCorpus, removePunct = TRUE)
seqs_token <- list(c('foreign', 'policy'), c('United', 'States'))
seqs_glob <- list(c('foreign', 'polic*'), c('United', 'States'))
seqs_regex <- list(c('^foreign', '^polic(ie|y)'), c('^United', '^States'))
toks2 <- joinTokens(toks, seqs_token, "_", 'fixed')
toks2 <- joinTokens(toks, seqs_glob, "_", 'glob')
toks2 <- joinTokens(toks, seqs_regex, "_", 'regex')
kwic(toks2, 'foreign_policy', window=1) # joined
kwic(toks2, c('foreign', 'policy'), window=1) # not joined
kwic(toks2, 'United_States', window=1) # joined
```

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

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

Value

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

Author(s)

Kenneth Benoit

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 *dfm with example data from Table 1 of Laver Benoit and Garry (2003)*

Description

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

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 *calculate lexical diversity*

Description

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

Usage

```
lexdiv(x, ...)

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

Arguments

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

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, $\log V_0$ & $\log {}_e V_0$):

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

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

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

Value

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

Note

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

Author(s)

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

References

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

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

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

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

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

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

Examples

```
mydfm <- dfm(subset(inaugCorpus, Year > 1980), verbose = FALSE)
(results <- lexdiv(mydfm, c("CTTR", "TTR", "U")))
cor(lexdiv(mydfm, "all"))

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

---

metacorpus *get or set corpus metadata*

Description

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

Usage

```
metacorpus(x, field = NULL)

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

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

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

Examples

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

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

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)
```

mobydickText *Project Gutenberg text of Herman Melville's* Moby Dick

Description

Named character object of the ASCII text of Herman Melville's *Moby Dick*, EBook no. 2701.

Source

Project Gutenberg, http://www.gutenberg.org

Examples

```
summary(mobydickText)
```

ndoc *get the number of documents or features*

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

x	a tokenizedText object or a character vector of tokens
...	not used
n	integer vector specifying the number of elements to be concatenated in each ngram

skip	integer vector specifying the adjacency skip size for tokens forming the ngrams, default is 0 for only immediately neighbouring words. For skipgrams, skip can be a vector of integers, as the "classic" approach to forming skip-grams is to set skip = k where k is the distance for which k or fewer skips are used to construct the n-gram. Thus a "4-skip-n-gram" defined as skip = 0:4 produces results that include 4 skips, 3 skips, 2 skips, 1 skip, and 0 skips (where 0 skips are typical n-grams formed from adjacent words). See Guthrie et al (2006).
concatenator	character for combining words, default is _ (underscore) character

Details

Normally, 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 *count the number of sentences*

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 tokenize

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 tokenize

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 *convert phrases into single tokens*

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, ...)

## S4 method for signature 'corpus,ANY'
phrasetotoken(object, phrases, ...)

## S4 method for signature 'character,dictionary'
phrasetotoken(object, phrases, ...)

## S4 method for signature 'character,collocations'
phrasetotoken(object, phrases, ...)

## S4 method for signature 'character,character'
phrasetotoken(object, phrases,
  concatenator = "_", valuetype = c("glob", "regex", "fixed"),
  case_insensitive = TRUE, ...)
```

Arguments

object	source texts, a character or character vector
phrases	a dictionary object that contains some phrases, defined as multiple words delimited by whitespace, up to 9 words long; or a quanteda collocation object created by collocations
...	additional arguments passed through to core "character,character" method
concatenator	the concatenation character that will connect the words making up the multi-word phrases. The default _ is highly recommended since it will not be removed during normal cleaning and tokenization (while nearly all other punctuation characters, at least those in the Unicode punctuation class [P] will be removed.
valuetype	how to interpret word matching patterns: "glob" for "glob"-style wildcarding, fixed for words as is; "regex" for regular expressions
case_insensitive	if TRUE, ignore case when matching

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 valuetype = "regex"
mydfm3 <- dfm(cw, dictionary = lapply(mydict, phrasetotoken, mydict),
              valuetype = "regex")
mydfm3
## one more token counted for "tax" than before
# using a dictionary to pre-process multi-word expressions
myDict <- dictionary(list(negative = c("bad* word*", "negative", "awful text"),
                          postiive = c("good stuff", "like? th??")))
txt <- c("I liked this, when we can use bad words, in awful text.",
         "Some damn good stuff, like the text, she likes that too.")
phrasetotoken(txt, myDict)

# on simple text
phrasetotoken("This is a simpler version of multi word expressions.", "multi word expression*")
```

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 comparison.cloud instead of a simple wordcloud, one grouping per document
...	additional parameters passed to to wordcloud or to text (and strheight, strwidth)

Details

The default is to plot the word cloud of all 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
```

```
docvars(ie2010Corpus, "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(govtoppDfm, comparison = TRUE)

## End(Not run)
```

`plot.kwic` *plot the dispersion of key word(s)*

Description

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

Usage

```
## S3 method for class 'kwic'
plot(..., scale = c("absolute", "relative"), sort = FALSE)
```

Arguments

`...`	any number of kwic class objects
`scale`	whether to scale the token index axis by absolute position of the token in the document or by relative position. Defaults are absolute for single document and relative for multiple documents.
`sort`	whether to sort the rows of a multiple document plot by document name

Value

`plot.kwic` returns a ggplot object

Author(s)

Adam Obeng

Examples

```
## Not run:
inaugCorpusPost70 <- subset(inaugCorpus, Year > 1970)
# compare multiple documents
plot(kwic(inaugCorpusPost70, "american"))
plot(kwic(inaugCorpusPost70, "american"), scale = "absolute")
```

```
# compare multiple terms across multiple documents
plot(kwic(inaugCorpusPost70, "america*"), kwic(inaugCorpusPost70, "people"))

# how to modify the ggplot with different options
library(ggplot2)
g <- plot(kwic(inaugCorpusPost70, "american"), kwic(inaugCorpusPost70, "people"))
g + aes(color = keyword) + scale_color_manual(values = c('red', 'blue'))

## End(Not run)
```

`predict.textmodel_NB_fitted` *prediction method for Naive Bayes classifier objects*

Description

implements class predictions using trained Naive Bayes examples

Usage

```
## S3 method for class 'textmodel_NB_fitted'
predict(object, newdata = NULL, ...)
```

Arguments

`object`	a fitted Naive Bayes textmodel
`newdata`	dfm on which prediction should be made
`...`	not used

Value

A list of two data frames, named `docs` and `words` corresponding to word- and document-level predicted quantities

`docs`	data frame with document-level predictive quantities: nb.predicted, ws.predicted, bs.predicted, PcGw, wordscore.doc, bayesscore.doc, posterior.diff, posterior.logdiff. Note that the diff quantities are currently implemented only for two-class solutions.
`words`	data-frame with word-level predictive quantities: wordscore.word, bayesscore.word

Author(s)

Kenneth Benoit

Examples

```
(nbfit <- textmodel_NB(LBGexample, c("A", "A", "B", "C", "C", NA)))
(nbpred <- predict(nbfit))
```

print.dfm *print a dfm object*

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

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

x	a tokenizedText object created by tokenize
...	further arguments passed to base print method

print.tokenSequences *print a tokenSequences objects*

Description

print method for a tokenSequences object

Usage

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

Arguments

x	a tokenSequences object created by findSequences
...	further arguments passed to base print method

`readability` *calculate readability*

Description

Calculate the readability of text(s).

Usage

```
readability(x, ...)

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

## 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.Liau.short", "Dale.Chall", "Dale.Chall.old",
  "Dale.Chall.PSK", "Danielson.Bryan", "Danielson.Bryan.2", "Dickes.Steiwer",
  "DRP", "ELF", "Farr.Jenkins.Paterson", "Flesch", "Flesch.PSK",
  "Flesch.Kincaid", "FOG", "FOG.PSK", "FOG.NRI", "FORCAST", "FORCAST.RGL",
  "Fucks", "Linsear.Write", "LIW", "nWS", "nWS.2", "nWS.3", "nWS.4", "RIX",
  "Scrabble", "SMOG", "SMOG.C", "SMOG.simple",      "SMOG.de", "Spache",
  "Spache.old", "Strain", "Traenkle.Bailer", "Traenkle.Bailer.2",
  "Wheeler.Smith", "meanSentenceLength", "meanWordSyllables"),
  removeHyphens = TRUE, drop = TRUE, ...)
```

Arguments

`x`	a corpus object or character vector
`...`	not used
`measure`	character vector defining the readability measure to calculate
`removeHyphens`	if `TRUE`, treat constituent words in hyphenated as separate terms, for purposes of computing word lengths, e.g. "decision-making" as two terms of lengths 8 and 6 characters respectively, rather than as a single word of 15 characters
`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), 2)
```

---

removeFeatures *remove features from an object*

Description

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. This function simply provides a convenience wrapper for `selectFeatures` where `selection = "remove"`.

Usage

```
removeFeatures(x, features, ...)
```

Arguments

`x`	object from which stopwords will be removed
`features`	character vector of features to remove
`...`	additional arguments passed to `selectFeatures`

Value

an object with matching features removed

Author(s)

Kenneth Benoit

See Also

stopwords

Examples

```
## for tokenized texts
txt <- c(wash1 <- "Fellow citizens, I am again called upon by the voice of my country to
                  execute the functions of its Chief Magistrate.",
       wash2 <- "When the occasion proper for it shall arrive, I shall endeavor to express
                  the high sense I entertain of this distinguished honor.")
removeFeatures(tokenize(txt, removePunct = TRUE), 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], n=20))
removeFeatures(myCollocs, stopwords("english"))
removeFeatures(myCollocs, stopwords("english"), pos = 2)
```

---

sample *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

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

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

## S3 method for class 'corpus'
sample(x, size = ndoc(x), 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 sample, which is not defined as a generic method in the **base** package.
what	dimension (of a dfm) 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

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 *compute the Scrabble letter values of text*

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)))), valuetype = c("regex", "fixed",
  "glob"), 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)))), valuetype = c("regex", "fixed",
  "glob"), perl = FALSE, keepdocvars = TRUE, ...)
```

Arguments

x	text or corpus object to be segmented
...	provides additional arguments passed to tokenize, if what = "tokens" is used
what	unit of segmentation. Current options are "tokens" (default), "sentences", "paragraphs", "tags", and "other". Segmenting on other allows segmentation of a text on any user-defined value, and must be accompanied by the delimiter argument. Segmenting on tags performs the same function but preserves the tags as a document variable in the segmented corpus.
delimiter	delimiter defined as a regex for segmentation. Each type has its own default, except other, which requires a value to be specified.
valuetype	how to interpret the delimiter: fixed for exact matching; "regex" for regular expressions; or "glob" for "glob"-style wildcard patterns
perl	logical. Should Perl-compatible regular expressions be used?
keepdocvars	if TRUE, repeat the docvar values for each segmented text; if FALSE, drop the docvars in the segmented corpus. Dropping the docvars might be useful in order to conserve space or if these are not desired for the segmented corpus.

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 correponding 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), segment(ukimmigTexts))

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

# segment a text into sentences
segmentedChar <- segment(ukimmigTexts, "sentences")
segmentedChar[2]
```

```
testCorpus <- corpus(c("##INTRO This is the introduction.
                       ##DOC1 This is the first document.
                       Second sentence in Doc 1.
                       ##DOC3 Third document starts here.
                       End of third document.",
                      "##INTRO Document ##NUMBER Two starts before ##NUMBER Three."))
# add a docvar
testCorpus[["serialno"]] <- paste0("textSerial", 1:ndoc(testCorpus))
testCorpusSeg <- segment(testCorpus, "tags")
summary(testCorpusSeg)
texts(testCorpusSeg)
# segment a corpus into sentences
segmentedCorpus <- segment(corpus(ukimmigTexts), "sentences")
identical(ndoc(segmentedCorpus), length(unlist(segmentedChar)))
```

`selectFeatures` *select features from an object*

Description

This function selects or discards features from a dfm.variety 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, selection = c("keep", "remove"),
  valuetype = c("glob", "regex", "fixed"), case_insensitive = TRUE,
  verbose = TRUE, ...)

## S3 method for class 'tokenizedTexts'
selectFeatures(x, features, selection = c("keep",
  "remove"), valuetype = c("glob", "regex", "fixed"),
  case_insensitive = TRUE, padding = FALSE, indexing = FALSE,
  verbose = FALSE, ...)

## S3 method for class 'collocations'
selectFeatures(x, features, selection = c("keep",
  "remove"), valuetype = c("glob", "regex", "fixed"),
  case_insensitive = TRUE, verbose = TRUE, pos = 1:3, ...)
```

Arguments

x	object whose features will be selected

features	one of: a character vector of features to be selected, a dfm whose features will be used for selection, or a dictionary class object whose values (not keys) will provide the features to be selected. For dfm objects, see details in the Value section below.
...	supplementary arguments passed to the underlying functions in stri_detect_regex. (This is how case_insensitive 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
padding	(only for tokenizedTexts objects) if TRUE, leave an empty string where the removed tokens previously existed. This is useful if a positional match is needed between the pre- and post-selected features, for instance if a window of adjacency needs to be computed.
indexing	use dfm-based index to efficiently process large tokenizedTexts object
pos	indexes of word position if called on collocations: remove if word pos is a stopword

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.")
(dfm1 <- dfm(textVec1, verbose = FALSE))
(dfm2a <- dfm(textVec2, verbose = FALSE))
(dfm2b <- selectFeatures(dfm2a, dfm1))
setequal(features(dfm1), features(dfm2b))

# more selection on a dfm
selectFeatures(dfm1, dfm2a)
selectFeatures(dfm1, dfm2a, selection = "remove")
## Not run: ## performance comparisons
data(SOTUCorpus, package = "quantedaData")
toks <- tokenize(SOTUCorpus, removePunct = TRUE)
# toks <- tokenize(tokenize(SOTUCorpus, what='sentence', simplify = TRUE), removePunct = TRUE)
# head to head, old v. new
system.time(selectFeaturesOLD(toks, stopwords("english"), "remove", verbose = FALSE))
system.time(selectFeatures(toks, stopwords("english"), "remove", verbose = FALSE))
system.time(selectFeaturesOLD(toks, c("and", "of"), "remove", verbose = FALSE, valuetype = "regex"))
system.time(selectFeatures(toks, c("and", "of"), "remove", verbose = FALSE, valuetype = "regex"))
microbenchmark::microbenchmark(
    old = selectFeaturesOLD(toks, stopwords("english"), "remove", verbose = FALSE),
    new = selectFeatures(toks, stopwords("english"), "remove", verbose = FALSE),
    times = 5, unit = "relative")
microbenchmark::microbenchmark(
   new = selectFeaturesOLD(toks, c("and", "of"), "remove", verbose = FALSE, valuetype = "regex"),
   old = selectFeatures(toks, c("and", "of"), "remove", verbose = FALSE, valuetype = "regex"),
    times = 2, unit = "relative")

types <- unique(unlist(toks))
numbers <- types[stringi::stri_detect_regex(types, '[0-9]')]
microbenchmark::microbenchmark(
   new = selectFeaturesOLD(toks, numbers, "remove", verbose = FALSE, valuetype = "fixed"),
    old = selectFeatures(toks, numbers, "remove", verbose = FALSE, valuetype = "fixed"),
    times = 2, unit = "relative")

# removing tokens before dfm, versus after
microbenchmark::microbenchmark(
    pre = dfm(selectFeaturesOLD(toks, stopwords("english"), "remove"), verbose = FALSE),
    post = dfm(toks, ignoredFeatures = stopwords("english"), verbose = FALSE),
    times = 5, unit = "relative")

## End(Not run)

## with simple examples
toks <- tokenize(c("This is a sentence.", "This is a second sentence."),
```

```
                removePunct = TRUE)
selectFeatures(toks, c("is", "a", "this"), selection = "remove",
               valuetype = "fixed", padding = TRUE, case_insensitive = TRUE)

# how case_insensitive works
selectFeatures(toks, c("is", "a", "this"), selection = "remove",
              valuetype = "fixed", padding = TRUE, case_insensitive = FALSE)
selectFeatures(toks, c("is", "a", "this"), selection = "remove",
              valuetype = "fixed", padding = TRUE, case_insensitive = TRUE)
selectFeatures(toks, c("is", "a", "this"), selection = "remove",
              valuetype = "glob", padding = TRUE, case_insensitive = TRUE)
selectFeatures(toks, c("is", "a", "this"), selection = "remove",
              valuetype = "glob", padding = TRUE, case_insensitive = FALSE)

# with longer texts
txts <- c(exampleString, inaugTexts[2])
toks <- tokenize(txts)
selectFeatures(toks, stopwords("english"), "remove")
selectFeatures(toks, stopwords("english"), "keep")
selectFeatures(toks, stopwords("english"), "remove", padding = TRUE)
selectFeatures(toks, stopwords("english"), "keep", padding = TRUE)
selectFeatures(tokenize(encodedTexts[1]), stopwords("english"), "remove", padding = TRUE)


## example for collocations
(myCollocs <- collocations(inaugTexts[1:3], n=20))
selectFeatures(myCollocs, stopwords("english"), "remove")
```

`selectFeaturesOLD` *old version of selectFeatures.tokenizedTexts*

Description

Calls C++ for super-fast selection or removal of features from a set of tokens.

Usage

```
selectFeaturesOLD(x, ...)

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

Arguments

`x`	object whose features will be selected
`...`	supplementary arguments passed to the underlying functions in `stri_detect_regex`. (This is how `case_insensitive` is passed, but you may wish to pass others.)

features	one of: a character vector of features to be selected, a dfm whose features will be used for selection, or a dictionary class object whose values (not keys) will provide the features to be selected. For dfm objects, see details in the Value section below.
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

Examples

```
toks <- tokenize(c("This is some example text from me.", "More of the example text."),
                 removePunct = TRUE)
selectFeaturesOLD(toks, stopwords("english"), "remove")
selectFeaturesOLD(toks, "ex", "keep", valuetype = "regex")
```

---

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 = NULL, n = NULL, margin = c("documents",
  "features"), method = "correlation", sorted = TRUE, normalize = FALSE)

## S4 method for signature 'dfm'
similarity(x, selection = NULL, n = NULL,
  margin = c("documents", "features"), method = "correlation",
  sorted = TRUE, normalize = FALSE)

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

## S3 method for class 'similMatrix'
print(x, digits = 4, ...)
```

Arguments

x	a dfm object
selection	character or character vector of document names or feature labels from the dfm
n	the top n most similar items will be returned, sorted in descending order. If n is NULL, return all items.
margin	identifies the margin of the dfm on which similarity will be computed: documents for documents or features for word/term features.
method	a valid method for computing similarity from pr_DB
sorted	sort results in descending order if TRUE
normalize	a deprecated argument retained (temporarily) for legacy reasons. If you want to compute similarity on a "normalized" dfm objects (e.g. x), wrap it in weight(x, "relFreq").
...	unused
digits	decimal places to display similarity values

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
(tmp <- similarity(presDfm, margin = "documents"))
# output as a matrix
as.matrix(tmp)
# for specific comparisons
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"), margin = "documents")
similarity(presDfm, c("2009-Obama" , "2013-Obama"), margin = "documents", method = "cosine")
similarity(presDfm, "2005-Bush", margin = "documents", method = "eJaccard", sorted = FALSE)

# compute some term similarities
similarity(presDfm, c("fair", "health", "terror"), method="cosine", margin = "features", 20)

## 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
quantedaDfm <- new("dfmSparse", Matrix::Matrix(t(as.matrix(tdm))))
similarity(quantedaDfm, c("oil", "opec", "xyz"), margin = "features", n = 14)
corMat <- as.matrix(proxy::simil(as.matrix(quantedaDfm), by_rows = FALSE))
round(head(sort(corMat[, "oil"], decreasing = TRUE), 14), 2)
round(head(sort(corMat[, "opec"], decreasing = TRUE), 9), 2)

## End(Not run)
```

`sort.dfm` *sort a dfm by one or more margins*

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 dfm
decreasing	TRUE (default) if sort will be in descending order, otherwise sort in increasing order
margin	which margin to sort on features to sort by frequency of features, docs to sort by total feature counts in documents, and both to sort by both
...	additional arguments passed to base method sort.int

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)
```

Arguments

kind	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
verbose	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/a11-smart-stop-list/english.stop) and a set of stopword lists from the Snowball stemmer project in different languages (see http://snowballstem.org/projects.html). Supported languages are arabic, danish, dutch, english, finnish, french, german, 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 encourged to inspect the list and to make sure it fits your particular requirements. The built-in English stopword list does not contain "will", for instance, because of its multiple meanings, but you might want to include this word for your own application.

Examples

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

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

---

subset.corpus *extract a subset of a corpus*

Description

Returns subsets of a corpus that meet certain conditions, including direct logical operations on docvars (document-level variables). 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 *summarize a corpus or a vector of texts*

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 desription of the texts, including number of types, tokens, and sentences.

Usage

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

## S3 method for class 'character'
summary(object, n = 100, verbose = TRUE,
  toLower = FALSE, ...)

describeTexts(object, n = 100, verbose = TRUE, ...)
```

Arguments

object	corpus or texts to be summarized
n	maximum number of texts to describe, default=100
verbose	set to FALSE to turn off printed output, for instance if you simply want to assign the output to a data.frame
showmeta	for a corpus, set to TRUE to include document-level meta-data
toLower	convert texts to lower case before counting types
...	additional arguments passed through to tokenize

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.  Second sentence.", "And this one."))
summary(ukimmigTexts)
myTextSummaryDF <- summary(ukimmigTexts, verbose = FALSE)
head(myTextSummaryDF)
```

---

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

x	character vector or tokenizedText-class object whose syllables will be counted
...	additional arguments passed to tokenize
syllableDict	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")
```

---

tail.tokenSequences *print a tokenSequences objects*

Description

print method for a tokenSequences object

Usage

```
## S3 method for class 'tokenSequences'
tail(x, ...)
```

Arguments

x	a tokenSequences object created by findSequences
...	further arguments passed to base print method

---

textfile *read a text corpus source from a file*

Description

Read a text corpus from one or more source files. The texts of the corpus come from (some part of) the content of the files, and the document-level metadata (docvars) come from either the file contents or filenames.

Usage

```
textfile(file, ignoreMissingFiles = FALSE, textField = NULL,
  cache = FALSE, docvarsfrom = c("filenames"), dvsep = "_",
  docvarnames = NULL, encoding = NULL, ...)

## S4 method for signature 'character'
textfile(file, ignoreMissingFiles = FALSE,
  textField = NULL, cache = FALSE, docvarsfrom = "metadata",
  dvsep = "_", docvarnames = NULL, encoding = NULL, ...)
```

Arguments

`file` the complete filename(s) to be read. This is designed to automagically handle a number of common scenarios, so the value can be a "glob"-type' wildcard value. Currently available filetypes are:

- `txt` plain text files: So-called structured text files, which describe both texts and metadata: For all structured text filetypes, the column, field, or node which contains the the text must be specified with the `textField` parameter, and all other fields are treated as docvars.
- `json` data in some form of JavaScript Object Notation, consisting of the texts and optionally additional docvars. The supported formats are:
  - a single JSON object per file
  - line-delimited JSON, with one object per line
  - line-delimited JSON, of the format produced from a Twitter stream. This type of file has special handling which simplifies the Twitter format into docvars. The correct format for each JSON file is automatically detected.

`csv,tab,tsv` comma- or tab-separated values

`xml` Basic flat XML documents are supported – those of the kind supported by the function xmlToDataFrame function of the **XML** package. `file` can also not be a path to a single local file, such as

a wildcard value any valid pathname with a wildcard ("glob") expression that can be expanded by the operating system. This may consist of multiple file types.

a URL to a remote which is downloaded then loaded

`zip,tar,tar.gz,tar.bz` archive file, which is unzipped. The contained files must be either at the top level or in a single directory. Archives, remote URLs and glob patterns can resolve to any of the other filetypes, so you could have, for example, a remote URL to a zip file which contained Twitter JSON files.

`ignoreMissingFiles`

if `FALSE`, then if the file argument doesn't resolve to an existing file, then an error will be thrown. Note that this can happen in a number of ways, including passing a path to a file that does not exist, to an empty archive file, or to a glob pattern that matches no files.

`textField` 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 `.csv` and `.json`.

`cache` If `TRUE`, write the object to a temporary file and store the temporary filename in the corpusSource-class 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 corpusSource-class object and the second as a corpus class object. It also provides a way to try different settings of encoding conversion when creating a corpus from a corpusSource-class object, without having to load in all of the source data again

`docvarsfrom` used to specify that docvars should be taken from the filenames, when the `textfile` inputs are filenames and the elements of the filenames are document variables, separated by a delimiter (`dvsep`). This allows easy assignment of docvars from filenames such as `1789-Washington.txt`, `1793-Washington`, etc. by `dvsep` or from meta-data embedded in the text file header (`headers`).

`dvsep` separator used in filenames to delimit docvar elements if `docvarsfrom="filenames"` is used

`docvarnames` character vector of variable names for `docvars`, if `docvarsfrom` is specified. If this argument is not used, default docvar names will be used (`docvar1`, `docvar2`, ...).

`encoding` vector: either the encoding of all files, or one encoding for each files

`...` additional arguments passed through to low-level file reading function, such as `file`, `read.csv`, etc. Useful for specifying an input encoding option, which is specified in the same was as it would be give to `iconv`. See the Encoding section of file for details. Also useful for passing arguments through to `read.csv`, for instance 'quote = ""', if quotes are causing problems within comma-delimited fields.

Details

If `cache = TRUE`, 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)

Adam Obeng, 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://wordpress.org/plugins/about/readme.txt")
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))

# vector of full filenames for a recursive structure
textfile(list.files(path = "~/Desktop/texts", pattern = "\\.txt$",
                    full.names = TRUE, recursive = TRUE))

## 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", "ca"), ...)

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

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

Arguments

`x`	a quanteda dfm object containing feature counts by document
`y`	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`.
`data`	dfm or data.frame from which to take the formula
`model`	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 textmodel_wordscores. `NB` Fits a Naive Bayes model to the dfm, with options for smoothing, setting class priors, and a choice of multinomial or binomial probabilities. See textmodel_NB. `wordfish` Fits the "wordfish" model of Slapin and Proksch (2008). See textmodel_wordfish. `ca` Correspondence analysis scaling of the dfm. `lda` Fit a topic model based on latent Dirichlet allocation. Not yet implemented – use `convert` to convert a dfm into the applicable input format and then use your favourite topic modelling package directly. `kNN` k-nearest neighbour classification, coming soon.
`...`	additional arguments to be passed to specific model types
`formula`	An object of class formula of the form `y ~ x1 + x2 + ...`. (Interactions are not currently allowed for any of the models implemented.) The `x` variable(s) is typically a dfm, 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 ca.matrix

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_NB *Naive Bayes classifier for texts*

Description

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

Usage

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

Arguments

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

Details

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

Value

A list of return values, consisting of:

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

Author(s)

Kenneth Benoit

Examples

```
## Example from 13.1 of _An Introduction to Information Retrieval_
trainingset <- as.dfm(matrix(c(1, 2, 0, 0, 0, 0,
                         0, 2, 0, 0, 1, 0,
                         0, 1, 0, 1, 0, 0,
                         0, 1, 1, 0, 0, 1,
                         0, 3, 1, 0, 0, 1),
                       ncol=6, nrow=5, byrow=TRUE,
                       dimnames = list(docs = paste("d", 1:5, sep = ""),
                                       features = c("Beijing", "Chinese",  "Japan", "Macao",
                                                    "Shanghai", "Tokyo"))))
trainingclass <- factor(c("Y", "Y", "Y", "N", NA), ordered = TRUE)
## replicate IIR p261 prediction for test set (document 5)
```

```
(nb.p261 <- textmodel_NB(trainingset, trainingclass))
predict(nb.p261, newdata = trainingset[5, ])

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

`textmodel_wordfish` *wordfish text model*

Description

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

Usage

```
textmodel_wordfish(data, dir = c(1, 2), priors = c(Inf, Inf, 3, 1),
  tol = c(1e-06, 1e-08), dispersion = c("poisson", "quasipoisson"),
  dispersionLevel = c("feature", "overall"), dispersionFloor = 0)

## 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`	prior precisions for the estimated parameters α_i, ψ_j, β_j, and θ_i, where i indexes documents and j indexes features
`tol`	tolerances for convergence. The first value is a convergence threshold for the log-posterior of the model, the second value is the tolerance in the difference in parameter values from the iterative conditional maximum likelihood (from conditionally estimating document-level, then feature-level parameters).
`dispersion`	sets whether a quasi-poisson quasi-likelihood should be used based on a single dispersion parameter (`"poisson"`), or quasi-Poisson (`"quasipoisson"`)
`dispersionLevel`	sets the unit level for the dispersion parameter, options are `"feature"` for term-level variances, or `"overall"` for a single dispersion parameter

dispersionFloor	constraint for the minimal underdispersion multiplier in the quasi-Poisson model. Used to minimize the distorting effect of terms with rare term or document frequencies that appear to be severely underdispersed. Default is 0, but this only applies if `dispersion = "quasipoisson"`.
x	for print method, the object to be printed
n	max rows of dfm to print
...	additional arguments passed to `print`
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 $\text{se}(\sigma) = 3$, through the third element in `priors`.

Value

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

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

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

Examples

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

## Not run:
ie2010dfm <- dfm(ie2010Corpus, verbose = FALSE)
(wfm1 <- textmodel_wordfish(ie2010dfm, dir = c(6,5)))
(wfm2a <- textmodel_wordfish(ie2010dfm, dir = c(6,5),
                             dispersion = "quasipoisson", dispersionFloor = 0))
(wfm2b <- textmodel_wordfish(ie2010dfm, dir = c(6,5),
                             dispersion = "quasipoisson", dispersionFloor = .5))
plot(wfm2a@phi, wfm2b@phi, xlab = "Min underdispersion = 0", ylab = "Min underdispersion = .5",
     xlim = c(0, 1.0), ylim = c(0, 1.0))
plot(wfm2a@phi, wfm2b@phi, xlab = "Min underdispersion = 0", ylab = "Min underdispersion = .5",
     xlim = c(0, 1.0), ylim = c(0, 1.0), type = "n")
underdispersedTerms <- sample(which(wfm2a@phi < 1.0), 5)
which(features(ie2010dfm) %in% names(topfeatures(ie2010dfm, 20)))
text(wfm2a@phi, wfm2b@phi, wfm2a@features,
     cex = .8, xlim = c(0, 1.0), ylim = c(0, 1.0), col = "grey90")
text(wfm2a@phi[underdispersedTerms], wfm2b@phi[underdispersedTerms],
     wfm2a@features[underdispersedTerms],
     cex = .8, xlim = c(0, 1.0), ylim = c(0, 1.0), col = "black")
if (require(austin)) {
    wfmodelAustin <- austin::wordfish(quanteda::as.wfm(ie2010dfm), dir = c(6,5))
    cor(wfm1@theta, wfm1Austin$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
smooth	a smoothing parameter for word counts; defaults to zero for the to match the LBG (2003) method.
object	a fitted Wordscores textmodel
newdata	dfm on which prediction should be made
rescaling	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.)
level	probability level for confidence interval width
verbose	If TRUE, output status messages
...	additional arguments passed to other functions
x	for print method, the object to be printed
n	max rows of dfm to print
digits	number of decimal places to print for print methods

Details

Fitting a textmodel_wordscores results in an object of class textmodel_wordscores_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 *get corpus texts*

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, ...)

texts(x) <- value

## S3 replacement method for class 'corpus'
texts(x) <- value

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

Arguments

x	A quanteda corpus object
groups	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 x is of type character, then groups must be a factor.
...	unused
value	character vector of the new texts

Value

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

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

Note

You are strongly encouraged as a good practice of text analysis workflow *not* to modify the substance of the texts in a corpus. Rather, this sort of processing is better performed through downstream operations. For instance, do not lowercase the texts in a corpus, or you will never be able to recover the original case. Rather, apply toLower to the corpus and use the result as an input, e.g. to tokenize.

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"])))

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

`tf` *compute (weighted) term frequency from a dfm*

Description

Apply varieties of term frequency weightings to a dfm.

Usage

```
tf(x, scheme = c("count", "prop", "propmax", "boolean", "log", "augmented",
  "logave"), base = 10, K = 0.5)

## S4 method for signature 'dfm'
tf(x, scheme = c("count", "prop", "propmax", "boolean", "log",
  "augmented", "logave"), base = 10, K = 0.5)
```

Arguments

`x`	object for which idf or tf-idf will be computed (a document-feature matrix)

`scheme`	divisor for the normalization of feature frequencies by document. Valid types include:
	`count` default, each feature count will remain as feature counts, equivalent to dividing by 1
	`prop` feature proportions within document, equivalent to dividing each term by the total count of features in the document.
	`propmax` feature proportions relative to the most frequent term of the document, equivalent to dividing term counts by the frequency of the most frequent term in the document.
	`boolean` recode all non-zero counts as 1
	`log` take the logarithm of 1 + each count, for base `base`
	`augmented` equivalent to K + (1 - K) * `tf(x, "propmax")`
	`logave` (1 + the log of the counts) / (1 + log of the counts / the average count within document)

`base`	base for the logarithm when `scheme` is `"log"` or `logave`
`K`	the K for the augmentation when `scheme = "augmented"`

Details

`tf(x, scheme = "prop")` is equivalent to `weight(x, "relFreq")`).

Value

A document feature matrix to which the weighting scheme has been applied.

Author(s)

Kenneth Benoit and Paul Nulty

References

Manning, C. D., Raghavan, P., & Schutze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

https://en.wikipedia.org/wiki/Tf-idf#Term_frequency_2

---

tfidf *compute tf-idf weights from a dfm*

Description

Compute tf-idf, inverse document frequency, and relative term frequency on document-feature matrices. See also `weight`.

Usage

```
tfidf(x, ...)

## S3 method for class 'dfm'
tfidf(x, normalize = FALSE, scheme = "inverse", ...)
```

Arguments

x	object for which idf or tf-idf will be computed (a document-feature matrix)
...	additional arguments passed to `docfreq` when calling `tfidf`
normalize	if `TRUE`, use relative term frequency
scheme	scheme for `docfreq`

Details

`tfidf` computes term frequency-inverse document frequency weighting. The default is not to normalize term frequency (by computing relative term frequency within document) but this will be performed if `normalize = TRUE`.

References

Manning, C. D., Raghavan, P., & Schutze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Examples

```
head(LBGexample[, 5:10])
head(tfidf(LBGexample)[, 5:10])
docfreq(LBGexample)[5:15]
head(tf(LBGexample)[, 5:10])

# replication of worked example from
# https://en.wikipedia.org/wiki/Tf-idf#Example_of_tf.E2.80.93idf
(wikiDfm <- new("dfmSparse",
                Matrix::Matrix(c(1,1,2,1,0,0, 1,1,0,0,2,3),
                   byrow = TRUE, nrow = 2,
                   dimnames = list(docs = c("document1", "document2"),
                     features = c("this", "is", "a", "sample", "another",
                                  "example")), sparse = TRUE)))
docfreq(wikiDfm)
tfidf(wikiDfm)
```

---

tokenize *tokenize a set of texts*

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,
  removeSymbols = FALSE, removeSeparators = TRUE, removeTwitter = FALSE,
  removeHyphens = FALSE, removeURL = FALSE, ngrams = 1L, skip = 0L,
  concatenator = "_", simplify = FALSE, verbose = FALSE, ...)

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

is.tokenizedTexts(x)

as.tokenizedTexts(x)
```

Arguments

x	The text(s) or corpus to be tokenized
...	additional arguments not used
what	the unit for splitting the text, available alternatives are: "word" (recommended default) smartest, but slowest, word tokenization method; see stringi-search-boundaries for details. "fasterword" dumber, but faster, word tokenizeation method, uses stri_split_charclass(x, "\\pWHITE_SPACE") "fastestword" dumbest, but fastest, word tokenization method, calls stri_split_fixed(x, " ") "character" tokenization into individual characters "sentence" sentence segmenter, smart enough to handle some exceptions in English such as "Prof. Plum killed Mrs. Peacock." (but far from perfect).
removeNumbers	remove tokens that consist only of numbers, but not words that start with digits, e.g. 2day
removePunct	if TRUE, remove all characters in the Unicode "Punctuation" [P] class
removeSymbols	if TRUE, remove all characters in the Unicode "Symbol" [S] class
removeSeparators	remove Separators and separator characters (spaces and variations of spaces, plus tab, newlines, and anything else in the Unicode "separator" category) when removePunct=FALSE. Only applicable for what = "character" (when you probably want it to be FALSE) and for what = "word" (when you probably want it to be TRUE). Note that if what = "word" and you set removePunct = TRUE, then removeSeparators has no effect. Use carefully.
removeTwitter	remove Twitter characters @ and #; set to TRUE if you wish to eliminate these.
removeHyphens	if TRUE, split words that are connected by hyphenation and hyphenation-like characters in between words, e.g. "self-storage" becomes c("self", "storage"). Default is FALSE to preserve such words as is, with the hyphens. Only applies if what = "word".
removeURL	if TRUE, find and eliminate URLs beginning with http(s) – see section "Dealing with URLs".
ngrams	integer vector of the *n* for *n*-grams, defaulting to 1 (unigrams). For bigrams, for instance, use 2; for bigrams and unigrams, use 1:2. You can even include irregular sequences such as 2:3 for bigrams and trigrams only. See ngrams.
skip	integer vector specifying the skips for skip-grams, default is 0 for only immediately neighbouring words. Only applies if ngrams is different from the default of 1. See skipgrams.
concatenator	character to use in concatenating *n*-grams, default is "_", which is recommended since this is included in the regular expression and Unicode definitions of "word" characters
simplify	if TRUE, return a character vector of tokens rather than a list of length ndoc(texts), with each element of the list containing a character vector of the tokens corresponding to that text.
verbose	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 by default from the text being tokenized except inter-word spacing and equivalent characters.

`as.tokenizedTexts` coerces a list of character tokens to a tokenizedText class object, making the methods available for this object type available to this object.

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.

Dealing with URLs

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

See the examples below.

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 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)
# keeping versus removing symbols
tokenize("<tags> and other + symbols.", removeSymbols = FALSE)
tokenize("<tags> and other + symbols.", removeSymbols = TRUE)
tokenize("<tags> and other + symbols.", removeSymbols = FALSE, what = "fasterword")
tokenize("<tags> and other + symbols.", removeSymbols = TRUE, what = "fasterword")

## examples with URLs - hardly perfect!
txt <- "Repo https://githib.com/kbenoit/quanteda, and www.stackoverflow.com."
tokenize(txt, removeURL = TRUE, removePunct = TRUE)
tokenize(txt, removeURL = FALSE, removePunct = TRUE)
tokenize(txt, removeURL = FALSE, removePunct = TRUE, what = "fasterword")
tokenize(txt, removeURL = FALSE, removePunct = FALSE, what = "fasterword")


## 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)
tokenize(txt, removeNumbers = TRUE, removePunct = TRUE, removeURL = TRUE)

# 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)
```

```
# removing features (stopwords) from tokenized texts
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"))

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

---

toLower *Convert texts to lower (or upper) case*

Description

Convert texts or tokens to lower (or upper) 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, ...)

toUpper(x, ...)

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

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

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

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

Arguments

x	texts to be lower-cased (or upper-cased)
keepAcronyms	if TRUE, do not lowercase any all-uppercase words. Only applies to toLower.
...	additional arguments passed to **stringi** functions, (e.g. stri_trans_tolower), such as locale

Value

Texts tranformed into their lower- (or upper-)cased versions. If x is a character vector or a corpus, return a character vector. If x is a list of tokenized texts, then return a list of 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)
test1 <- c(text1 = "England and France are members of NATO and UNESCO",
           text2 = "NASA sent a rocket into space.")
toUpper(test1)

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

---

topfeatures *list the most frequent features*

Description

List the most frequently occuring 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 *Trim a dfm using threshold-based or random feature selection*

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 dfm-class
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 stopword 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 occuring >=10 times and in >=2 docs
trim(myDfm, minCount = 10, minDoc = 2)
# only words occuring >=10 times and in at least 0.4 of the documents
trim(myDfm, minCount = 10, minDoc = 0.4)
# only words occuring at least 0.01 times and in >=2 documents
trim(myDfm, minCount = .01, minDoc = 2)
# only words occuring 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*

Description

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

Usage

```
weight(x, type, ...)

## S4 method for signature 'dfm,character'
weight(x, type = c("frequency", "relFreq",
  "relMaxFreq", "logFreq", "tfidf"), ...)

## S4 method for signature 'dfm,numeric'
weight(x, type, ...)

smoother(x, smoothing = 1)
```

Arguments

x	document-feature matrix created by dfm
type	a label of the weight type, or a named numeric vector of values to apply to the dfm. One of:
	"frequency" integer feature count (default when a dfm is created)

"relFreq" the proportion of the feature counts of total feature counts (aka relative frequency)

"relMaxFreq" the proportion of the feature counts of the highest feature count in a document

"logFreq" natural logarithm of the feature count

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

a named numeric vector a named numeric vector of weights to be applied to the dfm, where the names of the vector correspond to feature labels of the dfm, and the weights will be applied as multipliers to the existing feature counts for the corresponding named fatures. Any features not named will be assigned a weight of 1.0 (meaning they will be unchanged).

`...` not currently used. For finer grained control, consider calling tf or tfidf directly.

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

Details

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

Value

The dfm with weighted values.

Author(s)

Paul Nulty and Kenneth Benoit

References

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

See Also

tfidf

Examples

```
dtm <- dfm(inaugCorpus)
x <- apply(dtm, 1, function(tf) tf/max(tf))
topfeatures(dtm)
normDtm <- weight(dtm, "relFreq")
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
head(logTfDtm <- weight(dtm, type="logFreq"))
head(tfidf(logTfDtm, normalize = FALSE))

# apply numeric weights
str <- c("apple is better than banana", "banana banana apple much better")
weights <- c(apple = 5, banana = 3, much = 0.5)
(mydfm <- dfm(str, ignoredFeatures = stopwords("english"), verbose = FALSE))
weight(mydfm, weights)
```

`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 set of tokenized texts, whose word stems are to be removed. If tokenized texts, the tokenization must be word-based.
language	the name of a recognized language, as returned by getStemLanguages, or a two- or three-letter ISO-639 code corresponding to one of these languages (see references for the list of codes)

Value

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 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)*, 22
+,dfmSparse,numeric-method *(dfm-class)*, 22
+,numeric,dfmDense-method *(dfm-class)*, 22
+,numeric,dfmSparse-method *(dfm-class)*, 22
+.corpus *(corpus)*, 14
.stopwords *(stopwords)*, 71
[,dfmDense,index,index,logical-method *(dfm-class)*, 22
[,dfmDense,index,index,missing-method *(dfm-class)*, 22
[,dfmDense,index,missing,logical-method *(dfm-class)*, 22
[,dfmDense,index,missing,missing-method *(dfm-class)*, 22
[,dfmDense,logical,missing,missing-method *(dfm-class)*, 22
[,dfmDense,missing,index,logical-method *(dfm-class)*, 22
[,dfmDense,missing,index,missing-method *(dfm-class)*, 22
[,dfmDense,missing,missing,logical-method *(dfm-class)*, 22
[,dfmDense,missing,missing,missing-method *(dfm-class)*, 22
[,dfmSparse,index,index,logical-method *(dfm-class)*, 22
[,dfmSparse,index,index,missing-method *(dfm-class)*, 22
[,dfmSparse,index,missing,logical-method *(dfm-class)*, 22
[,dfmSparse,index,missing,missing-method *(dfm-class)*, 22
[,dfmSparse,logical,missing,missing-method *(dfm-class)*, 22
[,dfmSparse,missing,index,logical-method *(dfm-class)*, 22
[,dfmSparse,missing,index,missing-method *(dfm-class)*, 22
[,dfmSparse,missing,missing,logical-method *(dfm-class)*, 22
[,dfmSparse,missing,missing,missing-method *(dfm-class)*, 22
[.corpus, *30*
[.corpus *(corpus)*, 14
[[.corpus *(corpus)*, 14
[[<-.corpus *(corpus)*, 14

applyDictionary, 5, *20*
as.data.frame,dfm-method, 6
as.data.frame.dfm *(as.data.frame,dfm-method)*, 6
as.dfm *(dfm)*, 19
as.DocumentTermMatrix *(convert)*, 12
as.matrix,dfm-method *(dfm-class)*, 22
as.matrix.similMatrix *(similarity)*, 69
as.tokenizedTexts *(tokenize)*, 91
as.wfm *(convert)*, 12

c.corpus *(corpus)*, 14
ca.matrix, *80, 81*
cbind.dfm, 7, *11*
changeunits, 8, *62*
clean *(tokenize)*, 91
collocations, 9, *51*
colMeans,dfmSparse-method *(dfm-class)*, 22
colSums,dfmSparse-method *(dfm-class)*, 22
comparison.cloud, *52*
compress, 11
convert, 12, *79*
corpus, 14, *15*, *32*, *57*, *77, 78*
corpusSource-class, *15*, 18, *32*, *77, 78*

data.frame, *16*
describeTexts *(summary.corpus)*, 73

descriptive statistics on text, *5*
dfm, *4*, *7*, *11*, *13*, 19, *21*, *25*, *27*, *29*, *34*, *40*, *50*, *52*, *60*, *64*, *67*, *69–71*, *79, 80*, *89*, *93*, *96*, *99*
dfm-class, *6, 7*, *12*, *19*, *21*, 22, *27, 28*, *35, 36*, *64*, *81*, *97, 98*
dfm2ldaformat *(*convert*)*, 12
dfmDense-class *(*dfm-class*)*, 22
dfmSparse-class *(*dfm-class*)*, 22
dgCMatrix-class, *25*
dgeMatrix-class, *25*
dictionary, *5*, 26, *51*
docfreq, 27, *90*
docfreq,dfm-method *(*docfreq*)*, 27
docnames, *17*, 29
docnames<- *(*docnames*)*, 29
document-feature matrix, *41*
DocumentTermMatrix, *13*
docvars, *16, 17*, 30
docvars<- *(*docvars*)*, 30
drop, *16*

encodedTextFiles, 31
encodedTexts, 32
encoding, 32
englishSyllables *(*syllables*)*, 74
exampleString, 33

features, 34
file, *26*, *77*
findSequences, 34, *36*, *56*, *76*
formula, *79*

getStemLanguages, *102*

head,dfm-method *(*head.dfm*)*, 35
head.dfm, 35
head.tokenSequences, 36

iconv, *26*, *77*
ie2010Corpus, 37
iebudgets *(*ie2010Corpus*)*, 37
inaugCorpus, 37
inaugTexts *(*inaugCorpus*)*, 37
is.corpus *(*corpus*)*, 14
is.dfm *(*dfm*)*, 19
is.tokenizedTexts *(*tokenize*)*, 91

joinTokens, 38

key-words-in-context, *5*
kwic, 38, *53*

LBGexample, 40
lda.collapsed.gibbs.sampler, *13*
lexdiv, 40
lexical diversity measures, *5*

Matrix-class, *22*, *25*
metacorpus, *17*, 43
metacorpus<- *(*metacorpus*)*, 43
metadoc, *16, 17*, 44
metadoc<- *(*metadoc*)*, 44
mobydickText, 45

ndoc, *17*, 45, *92, 93*
nfeature *(*ndoc*)*, 45
ngrams, *11*, 46, *47*, *92, 93*
nsentence, 48
ntoken, *45*, 49
ntype *(*ntoken*)*, 49

phrasetotoken, 50
phrasetotoken,character,character-method *(*phrasetotoken*)*, 50
phrasetotoken,character,collocations-method *(*phrasetotoken*)*, 50
phrasetotoken,character,dictionary-method *(*phrasetotoken*)*, 50
phrasetotoken,corpus,ANY-method *(*phrasetotoken*)*, 50
plot.dfm, 52
plot.kwic, 53
pr_DB, *69*
predict.textmodel_NB_fitted, 54
predict.textmodel_wordscores_fitted *(*textmodel_wordscores*)*, 85
print, *84*
print,dfm-method *(*print.dfm*)*, 55
print,dfmDense-method *(*print.dfm*)*, 55
print,dfmSparse-method *(*print.dfm*)*, 55
print.dfm, 55
print.kwic *(*kwic*)*, 38
print.settings *(*settings*)*, 67
print.similMatrix *(*similarity*)*, 69
print.textmodel_wordfish_fitted *(*textmodel_wordfish*)*, 83
print.textmodel_wordscores_fitted *(*textmodel_wordscores*)*, 85

print.textmodel_wordscores_predicted (textmodel_wordscores), 85
print.tokenizedTexts, 56
print.tokenSequences, 56

quanteda (quanteda-package), 4
quanteda-package, 4, *37*
quantedaformat2dtm (convert), 12

rbind.dfm, *11*
rbind.dfm (cbind.dfm), 7
read.csv, *77*
readability, 57
readability indexes, *5*
regex, *62*
removeFeatures, 58, *64*
rowMeans,dfmSparse-method (dfm-class), 22
rowSums,dfmSparse-method (dfm-class), 22

sample, 59, *60*
scrabble, 60
segment, 61
select, *73*
selectFeatures, *20*, *58*, 63, *98*
selectFeaturesOLD, 66
settings, *17*, *25*, *55*, 67
settings<- (settings), 67
show,corpusSource-method (corpusSource-class), 18
show,dfmDense-method (print.dfm), 55
show,dfmSparse-method (print.dfm), 55
show,dictionary-method, 68
show,textmodel_wordfish_fitted-method (textmodel_wordfish), 83
show,textmodel_wordfish_predicted-method (textmodel_wordfish), 83
show,textmodel_wordscores_fitted-method (textmodel_wordscores), 85
show,textmodel_wordscores_predicted-method (textmodel_wordscores), 85
simil, *69*
similarities, *5*
similarity, 69
similarity,dfm-method (similarity), 69
skipgrams, *47*, *92*
skipgrams (ngrams), 46
smooth, *25*
smoother (weight), 99
sort.dfm, 70
stopwords, *20*, *58*, 71
strheight, *52*
stri_detect_regex, *64*, *66*
stri_enc_detect, *33*
stri_opts_brkiter, *48*
stri_split_charclass, *92*
stri_split_fixed, *92*
stri_trans_tolower, *96*
stringi-search-boundaries, *92*
strwidth, *52*
subset, *16*
subset.corpus, 72
summary.character (summary.corpus), 73
summary.corpus, 73
syllables, 74

t,dfmDense-method (dfm-class), 22
t,dfmSparse-method (dfm-class), 22
tail,dfm-method (head.dfm), 35
tail.dfm (head.dfm), 35
tail.tokenSequences, 75
TermDocumentMatrix, *14*
text, *52*
textfile, *15*, *32*, 76
textfile,character-method (textfile), 76
textmodel, 79, *80*, *85*
textmodel,dfm,ANY,missing,character-method (textmodel), 79
textmodel,formula,missing,dfm,character-method (textmodel), 79
textmodel_ca, 80
textmodel_fitted-class, *79*, 81
textmodel_NB, *79*, 81
textmodel_wordfish, *79*, 83
textmodel_wordfish_fitted-class (textmodel_fitted-class), 81
textmodel_wordfish_predicted-class (textmodel_fitted-class), 81
textmodel_wordscores, *79, 80*, 85
textmodel_wordscores_fitted-class (textmodel_fitted-class), 81
textmodel_wordscores_predicted-class (textmodel_fitted-class), 81
texts, *17*, 87
texts<- (texts), 87
tf, 89, *100*
tf,dfm-method (tf), 89
tfidf, 90, *100*

tokenise *(tokenize)*, 91
tokenize, *10*, *20*, *39*, *47–50*, *56*, *62*, *74*, *88*, 91
toLower, *88*, *93*, 95
topFeatures *(topfeatures)*, 96
topfeatures, 96
toUpper *(toLower)*, 95
trim, *64*, 97
trim,dfm-method *(trim)*, 97
trimdfm *(trim)*, 97

ukimmigTexts, 99
unlist, *43*

VCorpus, *15*, *17*

weight, *25*, *69*, *90*, 99
weight,dfm,character-method *(weight)*, 99
weight,dfm,numeric-method *(weight)*, 99
wordcloud, *52*
wordlists, 101
wordStem, *101, 102*
wordstem, 101