quanteda

March 7, 2015

Type Package

Title Quantitative Analysis of Textual Data

Version 0.7.1.002

Date 2015-03-07

Description A package for the management and quantitative analysis of textual data with R. 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 the texts in a corpus, for instance by tokenizine them, with or without stopwords or stemming, or to segment them by sentence or paragraph units. quanteda includes a suite of sophisticated tools to extract features of the texts into a quantitative matrix, where these features can be defined according to a dictionary or thesaurus, including the declaration of collocations to be treated as single features. Once converted into a quantitative matrix (known as a ``dfm" for document-feature matrix), the textual features can be analyzed using quantitative methods for describing, comparing, or scaling texts, or topic modelling, or used to train machine learning methods for class prediction.

License GPL-3

Depends R (>= 3.0)

Imports methods,
Matrix (>= 1.1),
data.table (>= 1.9.3),
SnowballC,
wordcloud,
proxy

LinkingTo Rcpp, RcppArmadillo

Suggests knitr,
lda,
topicmodels,
jsonlite (>= 0.9.10),
streamR,
tm,
slam

URL http://github.com/kbenoit/quanteda

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

LazyData TRUE

VignetteBuilder knitr

Collate 'RcppExports.R'
'clean.R'
'collocations.R'
'converters.R'
'corpus-sources-S4.R'
'corpus.R'
'dataDocs.R'
'describe-texts.R'
'dfm-classes.R'
'dfm-main.R'
'dictionaryFunctions.R'
'distance.R'
'gui.R'
'kwic.R'
'lexdiv.R'
'ngrams.R'
'onLoad.R'
'plots.R'
'quanteda-package.R'
'resample.R'
'settings.R'
'stopwords.R'
'syllables.R'
'textmodel-generics.R'
'textmodel-wordfish.R'
'textmodel-wordscores.R'
'tokenize.R'

R topics documented:

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

Description

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

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

Author(s)

Ken Benoit and Paul Nulty

bigrams *Create bigrams*

Description

Create bigrams

Usage

```
bigrams(text, window = 1, concatenator = "_", include.unigrams = FALSE,
  ignoredFeatures = NULL, skipGrams = FALSE, ...)
```

Arguments

`text`	character vector containing the texts from which bigrams will be constructed
`window`	how many words to be counted for adjacency. Default is 1 for only immediately neighbouring words. This is only available for bigrams, not for ngrams.
`concatenator`	character for combining words, default is _ (underscore) character
`include.unigrams`	if `TRUE`, return unigrams as well
`ignoredFeatures`	a character vector of features to ignore
`skipGrams`	If `FALSE` (default), remove any bigram containing a feature listed in `ignoredFeatures`, otherwise, first remove the features in `ignoredFeatures`, and then create bigrams. This means that some "bigrams" will actually not occur as adjacent features in the original text. See examples.
`...`	provides additional arguments passed to tokenize

Value

a character vector of bigrams

Author(s)

Ken Benoit and Kohei Watanabe

Examples

```
bigrams("The quick brown fox jumped over the lazy dog.")
bigrams(c("The quick brown fox", "jumped over the lazy dog."))
bigrams(c("The quick brown fox", "jumped over the lazy dog."), window=2)
bigrams(c("I went to tea with her majesty Queen Victoria.", "Does tea have extra caffeine?"))
bigrams(c("I went to tea with her majesty Queen Victoria.", "Does tea have extra caffeine?"),
        ignoredFeatures=stopwords("english"))
bigrams(c("I went to tea with her majesty Queen Victoria.", "Does tea have extra caffeine?"),
        ignoredFeatures=stopwords("english"), skipGrams=TRUE)
```

`changeunits` *change the document units of a corpus*

Description

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

Usage

```
changeunits(corp, to = c("sentences", "paragraphs", "documents"), ...)
```

Arguments

`corp`	corpus whose document units will be reshaped
`to`	new documents units for the corpus to be recast in
`...`	passes additional arguments to `segment`

Examples

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

# example with inaugural corpus speeches
mycorpus2 <- subset(inaugCorpus, Year>2004)
mycorpus2
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.
```

`clean` *simple cleaning of text before processing*

Description

`clean` removes punctuation and digits from text, using the regex character classes for punctuation and digits. `clean` uses the standard R function `tolower` to convert the text to lower case. Each of these steps is optional, but switched on by default, so for example, to remove punctuation and convert to lower, but keep digits, the command would be: `clean(mytexts, removeDigits=FALSE)`

Usage

```
clean(x, ...)

## S3 method for class 'character'
clean(x, removeDigits = TRUE, removePunct = TRUE,
  toLower = TRUE, removeAdditional = NULL, removeTwitter = FALSE,
  removeURL = TRUE, ...)

## S3 method for class 'corpus'
clean(x, removeDigits = TRUE, removePunct = TRUE,
  toLower = TRUE, removeAdditional = NULL, removeTwitter = FALSE, ...)
```

Arguments

`x`	The object to be cleaned. Can be either a character vector or a corpus object. If x is a corpus, `clean` returns the corpus containing the cleaned texts.
`...`	additional parameters
`removeDigits`	remove numbers if `TRUE`
`removePunct`	remove punctuation if `TRUE`
`toLower`	convert text to lower case `TRUE`
`removeAdditional`	additional characters to remove (regular expression)
`removeTwitter`	if `FALSE`, do not remove `@` or `#`
`removeURL`	removes URLs (web addresses starting with `http:` or `https:`), based on a regular expression from `http://daringfireball.net/2010/07/improved_regex_for_matching_urls`

Value

A character vector equal in length to the original texts, after cleaning.

Examples

```
clean("This is 1 sentence with 2.0 numbers in it, and one comma.", removeDigits=FALSE)
clean("This is 1 sentence with 2.0 numbers in it, and one comma.", toLower=FALSE)
clean("We are his Beliebers, and him is #ourjustin @justinbieber we luv u", removeTwitter=TRUE)
clean("Collocations can be represented as inheritance_tax using the _ character.")
clean("But under_scores can be removed with removeAdditional.", removeAdditional="[_]")
clean("This is a $1,500,000 budget and $20bn cash and a $5 cigar.")
clean("This is a $1,500,000 budget and $20bn cash and a $5 cigar.", removeDigits=FALSE)
clean("URL regex from http://daringfireball.net/2010/07/improved_regex_for_matching_urls.")

# for a vector of texts
clean(c("This is 1 sentence with 2.0 numbers in it, and one comma.",
        "$1.2 billion was spent on text analysis in 2014."))
```

`collocations` *Detect collocations from text*

Description

Detects collocations (currently, bigrams and trigrams) from texts or a corpus, returning a data.frame of collocations and their scores, sorted in descending order of the association measure.

Usage

```
collocations(x, ...)

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

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

Arguments

`x`	a text, a character vector of texts, or a corpus
`...`	additional parameters passed to clean
`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 $n_{11}/n_{1.} + n_{.1}$

`"all"` returns all of the above

`size`	length of the collocation. Only bigram (n=2) and trigram (n=3) collocations are implemented so far.
`n`	the number of collocations to return, sorted in descending order of the requested statistic, or G^2 if none is specified.

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

bigrams, ngrams

Examples

```
collocations(inaugTexts, n=10)
collocations(inaugCorpus, method="all", n=10)
collocations(inaugTexts, size=3, n=10)
collocations(inaugCorpus, method="all", size=3, n=10)
```

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

Description

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

Usage

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

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

as.wfm(x)

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

as.DocumentTermMatrix(x, ...)

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

dfm2ldaformat(x)
```

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

quantedaformat2dtm(x)

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

Arguments

`x`	dfm to be converted
`to`	target conversion format, consisting of the name of the package into whose document-term matrix representation the dfm will be converted: `"lda"` a list with components "documents" and "vocab" as needed by 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

Details

`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.

For individual converters in the foreign package idioms, return values are:

`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"))
## Not run: wftest <- austin::wordfish(austindfm)

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

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

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

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

`corpus` *constructor for corpus objects*

Description

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

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

```
## S3 method for class 'corpusSource'
corpus(x, enc = NULL, notes = NULL,
  citation = NULL, ...)

## S3 method for class 'VCorpus'
corpus(x, enc = NULL, notes = NULL, citation = NULL,
  ...)

is.corpus(x)

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

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
`enc`	A string specifying the input encoding for texts in the corpus. Must be a valid entry in `iconvlist()`, since the code in `corpus.character` will convert this to `UTF-8` using `iconv`. Currently only one input encoding can be specified for a collection of input texts, meaning that you should not mix input text encoding types in a single `corpus` call.
`docnames`	Names to be assigned to the texts, defaults to the names of the character vector (if any), otherwise assigns "text1", "text2", etc.
`docvars`	A data frame of attributes that is associated with each text.
`source`	A string specifying the source of the texts, used for referencing.
`notes`	A string containing notes about who created the text, warnings, To Dos, etc.
`citation`	Information on how to cite the corpus.
`c1`	corpus one to be added
`c2`	corpus two to be added

Details

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

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

Value

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

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

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

Note

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

Author(s)

Kenneth Benoit and Paul Nulty

See Also

docvars, metadoc, metacorpus, language, encoding, settings, texts

Examples

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

# create a corpus from texts and assign meta-data and document variables
ukimmigCorpus <- corpus(ukimmigTexts,
                            docvars=data.frame(party=names(ukimmigTexts)),
                            enc="UTF-8")
# the fifth column of this csv file is the text field
mytexts <- textfile("http://www.kenbenoit.net/files/text_example.csv", textField=5)
str(mytexts)
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))
#
## import a tm VCorpus
if (require(tm)) {
    data(crude)    # load in a tm example VCorpus
    mytmCorpus <- corpus(crude)
    summary(mytmCorpus, showmeta=TRUE)
}
```

`corpusSource-class` *corpus source classes*

Description

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

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

`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, clean = TRUE, stem = FALSE,
  ignoredFeatures = NULL, keptFeatures = NULL, matrixType = c("sparse",
  "dense"), language = "english", fromCorpus = FALSE, bigrams = FALSE,
  thesaurus = NULL, dictionary = NULL, dictionary_regex = FALSE,
  addto = NULL, ...)

## S3 method for class 'corpus'
dfm(x, verbose = TRUE, clean = TRUE, stem = FALSE,
  ignoredFeatures = NULL, keptFeatures = NULL, matrixType = c("sparse",
  "dense"), language = "english", groups = NULL, bigrams = FALSE,
  thesaurus = NULL, dictionary = NULL, dictionary_regex = FALSE,
  addto = 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 `clean`
`verbose`	display messages if `TRUE`
`clean`	if `FALSE`, do no cleaning of the text. This offers a one-argument easy method to turn off any cleaning of the texts during construction of the dfm.
`stem`	if `TRUE`, stem words
`ignoredFeatures`	a character vector of user-supplied features to ignore, such as "stop words". Formerly, this was a Boolean option for `stopwords = TRUE`, but requiring the user to supply the list highlights the choice involved in using any stopword list. To access one possible list (from any list you wish), use `stopwords()`.
`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 = "^@\\w+\\b"` and make sure that `removeTwitter = FALSE` as an additional argument passed to clean. (Note: `keptFeatures = "^@"` will also retrieve usernames, but does not enforce the username convention that a username must contain one and only one `@` symbol, at the beginning of the username.)
`matrixType`	if `dense`, produce a dense matrix; or it `sparse` produce a sparse matrix of class `dgCMatrix` from the **Matrix** package.
`language`	Language for stemming and stopwords. Choices are `danish`, `dutch`, `english`, `finnish`, `french`, `german`, `hungarian`, `italian`, `norwegian`, `porter`, `portuguese`, `romanian`, `russian`, `spanish`, `swedish`, `turkish` for stemming, and `SMART`, `danish`, `english`, `french`, `hungarian`, `norwegian`, `russian`, `swedish`, `catalan`, `dutch`, `finnish`, `german`, `italian`, `portuguese`, `spanish`, `arabic` for stopwords.
`fromCorpus`	a system flag used internally, soon to be phased out.
`bigrams`	include bigrams as well as unigram features, if `TRUE`
`thesaurus`	A list of character vector "thesaurus" entries, in a dictionary list format, which can also include regular expressions if `dictionary_regex` is `TRUE` (see examples). Note that unlike dictionaries, each entry in a thesaurus key must be unique, otherwise only the first match in the list will be used. Thesaurus keys are converted to upper case to create a feature label in the dfm, as a reminder that this was not a type found in the text, but rather the label of a thesaurus key.
`dictionary`	A list of character vector dictionary entries, including regular expressions (see examples)
`dictionary_regex`	`TRUE` means the dictionary is already in regular expression format, otherwise it will be converted from "wildcard" format
`addto`	`NULL` by default, but if an existing dfm object is specified, then the new dfm will be added to the one named. If both dfm's are built from dictionaries, the combined dfm will have its `Non_Dictionary` total adjusted.
`groups`	Grouping variable for aggregating documents

Details

New as of v0.7: All dfms are by default sparse, a change from the previous behaviour. You can still create the older (S3) dense matrix type dfm object, but you will receive a disapproving warning message while doing so, suggesting you make the switch.

`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.

If you used `matrixType = "dense"` then the return is an old-style S3 matrix class object with additional attributes representing meta-data.

Author(s)

Kenneth Benoit

Examples

```
# with inaugural texts
(size1 <- object.size(dfm(inaugTexts, matrixType="sparse")))
(size2 <- object.size(dfm(inaugTexts, matrixType="dense")))
cat("Compacted by ", round(as.numeric((1-size1/size2)*100), 1), "%.\n", sep="")

# for a corpus
mydfm <- dfm(inaugCorpus)

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

# with stopwords English, stemming, and dense matrix
dfmsInaug2 <- dfm(inaugCorpus, ignoredFeatures = stopwordsGet(), stem=TRUE, matrixType="dense")

## 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 <- 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"), dictionary_regex=TRUE)

## removing stopwords
testText <- "The quick brown fox named Seamus jumps over the lazy dog also named Seamus, with
```

```
             the newspaper from a a boy named Seamus, in his mouth."
testCorpus <- corpus(testText)
settings(testCorpus, "stopwords")
dfm(testCorpus, stopwords=TRUE)

## keep only certain words
dfm(testCorpus, keptFeatures="s$", verbose=FALSE)  # keep only words ending in "s"
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="^#")  # keep only hashtags


## Not run:
# try it with approx 35,000 court documents from Lauderdale and Clark (200?)
load('~/Dropbox/QUANTESS/Manuscripts/Collocations/Corpora/lauderdaleClark/Opinion_files.RData')
txts <- unlist(Opinion_files[1])
names(txts) <- NULL

# dfms without cleaning
require(Matrix)
system.time(dfmsBig <- dfm(txts, clean=FALSE, verbose=FALSE))
object.size(dfmsBig)
dim(dfmsBig)
# compare with tm
require(tm)
tmcorp <- VCorpus(VectorSource(txts))
system.time(tmDTM <- DocumentTermMatrix(tmcorp))
object.size(tmDTM)
dim(tmDTM)

# with cleaning - the gsub() calls in clean() take a long time
system.time(dfmsBig <- dfm(txts, clean=TRUE, additional="[-_\\x{h2014}]"))
object.size(dfmsBig)
dim(dfmsBig)
# 100 top features
topf <- colSums(dfmsBig)
names(topf) <- colnames(dfmsBig)
head(sort(topf, decreasing=TRUE), 100)

## End(Not run)
```

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

Description

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

Usage

```
## S4 method for signature 'dfm'
t(x)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Arguments

`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

`dictionary` *create a dictionary*

Description

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

Usage

```
dictionary(x = NULL, file = NULL, format = NULL, enc = "",
  tolower = TRUE, maxcats = 10)
```

Arguments

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

Value

A list with a ductionary class label, to be used by other functions in quanteda.

Note

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

References

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

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

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"))
dfm(mycorpus, dictionary=mydict)
## Not run:
# import the Laver-Garry dictionary from http://bit.ly/1FH2nvf
lgdict <- dictionary(file="~/Dropbox/QUANTESS/dictionaries/Misc Provalis/LaverGarry.cat",
                     format="wordstat")
dfm(inaugTexts, dictionary=lgdict)

# import a LIWC formatted dictionary
liwcdict <- dictionary(file = "~/Dropbox/QUANTESS/dictionaries/LIWC/LIWC2001_English.dic",
                        format = "LIWC")
dfm(inaugTexts, dictionary=liwcdict)

## End(Not run)
```

docfreq *get the document frequency of a feature*

Description

Returns the document frequency of a feature in a dfm-class object, which occurs greater than a threshold.

Usage

```
docfreq(object, threshold = 0)

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

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

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

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

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

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

Arguments

`object`	a dfm-class document-feature matrix
`threshold`	numeric value of the threshold for counting a feature as existing in the document, default is 0

docnames *get or set document names*

Description

Extract the document names from a corpus or a document-feature matrix. Document names are the rownames of the documents data.frame in a corpus, or the rownames of the dfm object for a dfm. of the dfm object.

docnames queries the document names of a corpus or a dfm

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

Usage

```
docnames(x)

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

docnames(x) <- value

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

Arguments

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

Value

docnames returns a character vector of the document names

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

Examples

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

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

---

docvars *get or set for document-level variables*

Description

Get or set variables for the documents in a corpus

Usage

```
docvars(x, field = NULL)

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

Arguments

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

Value

`docvars` returns a data.frame of the document-level variables

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

Examples

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

`encoding` *get the encoding of documents in a corpus*

Description

Get or set the `_encoding` document-level metadata field(s) in a corpus.

Usage

```
encoding(x, drop = TRUE)

encoding(x) <- value
```

Arguments

`x`	a corpus object
`drop`	return as a vector if `TRUE`, otherwise return a `data.frame`
`value`	a character vector or scalar representing the new value of the encoding (see Note)

Details

This function modifies the `_encoding` value set by `metadoc`. It is a wrapper for `metadoc(corp, "encoding")`.

Note

This function differs from R's built-in Encoding function, which only allows the four values of "latin1", "UTF-8", "bytes", and "unknown" (and which assigns "unknown" to any text that contains only ASCII characters). Legal values for encodings must be from iconvlist. Note that encoding does not convert or set encodings, it simply records a user declaration of a valid encoding. (We hope to implement checking and conversion later.)

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

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

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

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

Description

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

Usage

```
kwic(x, word, window = 5, wholeword = FALSE)

## S3 method for class 'character'
kwic(x, word, window = 5, wholeword = FALSE)

## S3 method for class 'corpus'
kwic(x, word, window = 5, wholeword = FALSE)
```

Arguments

`x`	A text character scalar or a quanteda corpus. (Currently does not support character vectors.)
`word`	A keyword chosen by the user.
`window`	The number of context words to be displayed around the keyword.
`wholeword`	If `TRUE`, then only search for the entire "word". Otherwise `word` is interpreted as a regular expression, which matches any occurrence of `word` in the text, so that the the concordance will include all words in which the search term appears, and not just when it appears as an entire word. For instance, searching for the word "key" will also return "whiskey". This is the default.

Value

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

Author(s)

Kenneth Benoit and Paul Nulty

Examples

```
kwic(inaugTexts, "terror")
kwic(inaugTexts, "terror", wholeword=TRUE)  # returns only whole word, without trailing punctuation
```

`language` *get or set the language of corpus documents*

Description

Get or set the `_language` document-level metadata field in a corpus.

Usage

```
language(corp, drop = TRUE)

language(corp) <- value
```

Arguments

`corp`	a corpus object
`drop`	return as a vector if `TRUE`, otherwise return a `data.frame`
`value`	the new value for the language meta-data field, a string or character vector equal in length to `ndoc(corp)`

Details

This function modifies the `_language` value set by `metadoc`. It is a wrapper for `metadoc(corp, "language")`.

`LBGexample` *dfm containing example data from Table 1 of Laver Benoit and Garry (2003)*

Description

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

Format

A dfm object with 6 documents and 37 features

Details

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

References

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

`lexdiv` *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("TTR", "C", "R", "CTTR", "U", "S",
  "Maas"), log.base = 10, ...)

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

Arguments

`x`	a document-feature matrix object
`...`	additional arguments
`measure`	A character vector defining the measure to calculate.
`log.base`	A numeric value defining the base of the logarithm.

Details

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

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

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

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

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

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

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

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

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

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

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

"S": Summer's index:

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

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

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

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

"Maas": Maas' indices (a, $\log V_0$ & $\log {}_e V_0$):

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

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

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

Note

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

Author(s)

Kenneth Benoit, adapted from the S4 class implementation written by Meik Michalke in the **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(inaugCorpus)
mydfmSW <- dfm(inaugCorpus, stopwords=TRUE)
results <- data.frame(TTR = lexdiv(mydfm, "TTR"),
                      CTTR = lexdiv(mydfm, "CTTR"),
                      U = lexdiv(mydfm, "U"),
                      TTRs = lexdiv(mydfmSW, "TTR"),
                      CTTRs = lexdiv(mydfmSW, "CTTR"),
                      Us = lexdiv(mydfmSW, "U"))
results
cor(results)
t(lexdiv(mydfmSW, "Maas"))
```

`metacorpus` *get or set corpus metadata*

Description

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

Usage

```
metacorpus(corp, field = NULL)

metacorpus(corp, field) <- value
```

Arguments

`corp`	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(corp, field = NULL)

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

Arguments

`corp`	A quanteda corpus object
`field`	string containing 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 `_encoding`, but when named in in the `field` argument, do *not* need the underscore character.

Examples

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

`ndoc` *get the number of documents or features*

Description

Returns the number of documents or features in a quanteda object.

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
`...`	additional parameters

Value

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

Examples

```
ndoc(inaugCorpus)
ndoc(dfm(inaugCorpus))
nfeature(dfm(inaugCorpus))
nfeature(trim(dfm(inaugCorpus), minDoc=5, minCount=10))
```

`ngrams` *Create ngrams*

Description

Create a set of ngrams (words in sequence) from text(s) in a character vector

Usage

```
ngrams(text, n = 2, concatenator = "_", include.all = FALSE, ...)
```

Arguments

`text`	character vector containing the texts from which ngrams will be extracted
`n`	the number of tokens to concatenate. Default is 2 for bigrams.
`concatenator`	character for combining words, default is _ (underscore) character
`include.all`	if TRUE, add n-1...1 grams to the returned list
`...`	additional parameters

Details

... provides additional arguments passed to tokenize

Value

a list of character vectors of ngrams, one list element per text

Author(s)

Ken Benoit, Kohei Watanabe, Paul Nulty

Examples

```
ngrams("The quick brown fox jumped over the lazy dog.", n=2)
identical(ngrams("The quick brown fox jumped over the lazy dog.", n=2),
          bigrams("The quick brown fox jumped over the lazy dog.", n=2))
ngrams("The quick brown fox jumped over the lazy dog.", n=3)
ngrams("The quick brown fox jumped over the lazy dog.", n=3, concatenator="~")
ngrams("The quick brown fox jumped over the lazy dog.", n=3, include.all=TRUE)
```

`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(x, dictionary, concatenator = "_")

## S3 method for class 'character'
phrasetotoken(x, dictionary, concatenator = "_")
```

Arguments

`x`	source texts, a character or character vector
`dictionary`	a list or named list (such as a quanteda dictionary) that contains some phrases, defined as multiple words delimited by whitespace. These can be up to 9 words long.
`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 POSIX class `[:punct:]`) will be removed.

Value

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

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 <- list(tax=c("tax", "income tax", "capital gains tax", "inheritance tax"))
(cw <- phrasetotoken(mytexts, mydict))
dfm(cw, verbose=FALSE)

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

`plot.dfm` *plot features as a wordcloud*

Description

The default plot method for a `dfm` object. Produces a wordcloud plot for the features of the dfm, weighted by the total frequencies. To produce word cloud plots for specific documents, the only way currently to do this is to produce a dfm only from the documents whose features you want plotted.

Usage

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

Arguments

`x`	a dfm object
`...`	additional parameters passed to to wordcloud or to text (and strheight, strwidth)

See Also

wordcloud

Examples

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

# plot only Lincoln's inaugural address
plot(dfm(subset(inaugCorpus, President=="Lincoln"), verbose=FALSE, stopwords=TRUE))

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

print.dfm *print a dfm object*

Description

print methods for document-feature matrices

Usage

```
## S4 method for signature 'dfmSparse'
print(x, show.values = FALSE, show.settings = FALSE,
  ...)

## S4 method for signature 'dfmDense'
print(x, show.values = FALSE, show.settings = FALSE,
  ...)

## 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, ...)
```

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.
`...`	further arguments passed to or from other methods
`object`	the item to be printed

removeFeatures *remove features from an object*

Description

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

Usage

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

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

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

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

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

Arguments

`x`	object from which stopwords will be removed
`stopwords`	character vector of features to remove. Now requires an explicit list to be supplied, for instance `stopwords("english")`.
`verbose`	if `TRUE` print message about how many features were removed

Details

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

Value

an object with stopwords removed

See Also

stopwords

Examples

```
## examples for character objects
someText <- "Here's some text containing words we want to remove."
removeFeatures(someText, stopwords("english", verbose=FALSE))
removeFeatures(someText, stopwords("SMART", verbose=FALSE))
removeFeatures(someText, c("some", "want"))
itText <- "Ecco alcuni di testo contenente le parole che vogliamo rimuovere."
removeFeatures(itText, stopwords("italian", verbose=FALSE))

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

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

segment *segment texts into component elements*

Description

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

Usage

```
segment(x, ...)

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

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

Arguments

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

Details

Tokens are delimited by whitespace. 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, lower=FALSE), segment(ukimmigTexts, lower=FALSE))

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

# segment a text into sentences
segmentedChar <- segment(ukimmigTexts, "sentences")
segmentedChar[2]
testCorpus <- corpus("##INTRO This is the introduction.
                      ##DOC1 This is the first document.
                      Second sentence in Doc 1.
                      ##DOC3 Third document starts here.
                      End of third document.")
testCorpusSeg <- segment(testCorpus, "tags")
summary(testCorpusSeg)
texts(testCorpusSeg)
# segment a corpus into sentences
segmentedCorpus <- segment(corpus(ukimmigTexts), "sentences")
identical(ndoc(segmentedCorpus), length(unlist(segmentedChar)))
```

`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(inaugCorpus)
tempdfmSW <- dfm(inaugCorpus, stopwords=TRUE)
settings(inaugCorpus, "stopwords") <- TRUE
tempdfmSW <- dfm(inaugCorpus)
tempdfm <- dfm(inaugCorpus, stem=TRUE)
settings(tempdfm)
```

`similarity` *compute similarities between documents and/or features*

Description

Compute similarities between documents and/or features from a `dfm`. Uses the similarity measures defined in simil. See `pr_DB` for available distance measures, or how to create your own.

Usage

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

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

Arguments

`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: `features` for word/term features or `documents` for documents.
method	a valid method for computing similarity from `pr_DB`
sort	sort results in descending order if `TRUE`
normalize	if `TRUE`, normalize the dfm by term frequency within document (so that the dfm values will be relative term frequency within each document)
digits	digits for rounding results

Value

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

Note

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

Examples

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

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

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

## Not run:

# compare to tm
require(tm)
data("crude")
crude <- tm_map(crude, content_transformer(tolower))
crude <- tm_map(crude, removePunctuation)
crude <- tm_map(crude, removeNumbers)
crude <- tm_map(crude, stemDocument)
tdm <- TermDocumentMatrix(crude)
findAssocs(tdm, c("oil", "opec", "xyz"), c(0.75, 0.82, 0.1))
# in quanteda
crudeDfm <- dfm(corpus(crude))
similarity(crudeDfm, c("oil", "opec", "xyz"), normalize=FALSE, digits=2)

## End(Not run)
```

sort.dfm *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 = TRUE)

stopwordsGet(kind = "english")
```

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 (obtained from http://svn.tartarus.org/snowball/trunk/website/algorithms/*/stop.txt). Supported languages are arabic, danish, dutch, english, finnish, french, german, hungarian, italian, norwegian, portuguese, russian, spanish, and swedish. Language names are case sensitive.

Value

a character vector of stopwords

A note of caution

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

Examples

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

subset.corpus *extract a subset of a corpus*

Description

Works just like the normal subset command but for corpus objects

Usage

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

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
...	additional arguments affecting the summary produced

Value

corpus object

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, ...)

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

describeTexts(object, 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
`...`	additional arguments affecting the summary produced

Examples

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

`syllables` *count syllables in a text*

Description

This function takes a text and returns a count of the number of syllables it contains. For British 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 = englishSyllables, ...)
```

Arguments

`x`	character vector or list of character vectors whose syllables will be counted
`...`	additional arguments passed to clean
`syllableDict`	a named numeric vector of syllable counts where the names are lower case tokens. The default is englishSyllables, an English pronunciation dictionary from CMU.

Value

numeric Named vector or list of counts of the number of syllables for each element of x. When a word is not available in the lookup table, its syllables are estimated by counting the number of (English) vowels in the word.

Source

englishSyllables is built from the freely available CMU pronunciation dictionary at http://www.speech.cs.cmu.edu/cgi-bin/cmudict.

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("supercalifragilisticexpialidocious")
```

---

textfile *read a text corpus source from a file*

Description

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

Usage

```
textfile(file, textField, directory = NULL, docvarsfrom = c("filenames"),
  sep = "_", docvarnames = NULL, ...)

## S4 method for signature 'character,index,missing,missing,missing,missing'
textfile(file,
  textField, directory = NULL, docvarsfrom = c("filenames"), sep = "_",
  docvarnames = NULL, ...)

## S4 method for signature 'character,missing,missing,missing,missing,missing'
textfile(file,
  textField, directory = NULL, docvarsfrom = c("filenames"), sep = "_",
  docvarnames = NULL, ...)

## S4 method for signature 'character,missing,missing,character,ANY,ANY'
textfile(file,
  textField = NULL, directory = NULL, docvarsfrom = c("headers"),
  sep = "_", docvarnames = NULL, ...)
```

Arguments

`file`	the complete filename to be read. Currently available file types are: `txt` plain text files `json` data in JavaScript Object Notation, consisting of the texts and additional document-level variables and document-level meta-data. The text key must be identified by specifying a `textField` value. `csv` comma separated value data, consisting of the texts and additional document-level variables and document-level meta-data. The text file must be identified by specifying a `textField` value. **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. `doc, docx`**:** Word files coming soon. `pdf`**:** Adobe Portable Document Format files, coming soon.
`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`.
`directory`	not used yet, and may be removed (if I move this to a new method called `textfiles`)
`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 (`sep`). This allows easy assignment of docvars from filenames such as `1789-Washington.txt`, `1793-Washington`, etc. by `sep` or from meta-data embedded in the text file header (`headers`).
`sep`	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`, ...).
`...`	additional arguments passed through to other functions

Details

The constructor does not store a copy of the texts, but rather reads in the texts and associated data, and saves them to a temporary R object whose location is specified in the corpusSource-class object. This prevents a complete copy of the object from cluttering the global environment and consuming additional space. This does mean however that the state of the file containing the source data will not be cross-platform and may not be persistent across sessions. So the recommended usage is to load the data into a corpus in the same session in which `textfile` is called.

Value

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

Author(s)

Kenneth Benoit and Paul Nulty

Examples

```
# Twitter json
mytf <- textfile("~/Dropbox/QUANTESS/corpora/misc/NinTANDO_Me.json")
summary(corpus(mytf))
# generic json - needs a textField specifier
mytf2 <- textfile("~/Dropbox/QUANTESS/Manuscripts/Collocations/Corpora/sotu/sotu.json",
                  textField = "text")
summary(corpus(mytf2))
# text file
mytf3 <- textfile("~/Dropbox/QUANTESS/corpora/project_gutenberg/pg2701.txt")
summary(corpus(mytf3))
mytf4 <- textfile("~/Dropbox/QUANTESS/corpora/inaugural/*.txt")
summary(corpus(mytf4))
mytf5 <- textfile("~/Dropbox/QUANTESS/corpora/inaugural/*.txt",
                  docvarsfrom="filenames", sep="-", docvarnames=c("Year", "President"))
summary(corpus(mytf5))
```

textmodel *fit a text model*

Description

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

Usage

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

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

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

Arguments

`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.

`ca` Correspondence analysis scaling of the dfm.

NB Fits a Naive Bayes model to the dfm, with options for smoothing, setting class priors, and a choice of multinomial or binomial probabilities.

wordfish Fits the "wordfish" model of Slapin and Proksch (2008).

lda Fit a topic model based on latent Dirichlet allocation. Temporarily removed.

kNN k-nearest neighbour classification, coming soon.

`...`	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

```
## Not run:
data(ie2010Corpus, package="quantedaData")
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")

wf <- textmodel(ieDfm, model="wordfish", dir = c(6,5))
wf

## End(Not run)
```

textmodel_fitted-class *the fitted textmodel classes*

Description

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

Details

Available types currently include...

Slots

`dfm` a dfm-class document-feature matrix

textmodel_wordfish *wordfish text model*

Description

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

Usage

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

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

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

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

Arguments

`data`	the dfm on which the model will be fit
`dir`	set global identification by specifying the indexes for a pair of documents such that $\hat{\theta}_{dir[1]} < \hat{\theta}_{dir[2]}$.
`priors`	priors for θ_i, α_i, ψ_j, and β_j where i indexes documents and j indexes features
`tol`	tolerances for convergence (explain why a pair)
`x`	for print method, the object to be printed
`n`	max rows of dfm to print
`...`	additional arguments passed to other functions
`object`	wordfish fitted or predicted object to be shown

Details

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

Value

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

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

Author(s)

Benjamin Lauderdale and Kenneth Benoit

References

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

Examples

```
if (require(quantedaData)) {
    data(ie2010Corpus)
    ie2010dfm <- dfm(ie2010Corpus, verbose=FALSE)
    wfmodel <- textmodel_wordfish(ie2010dfm, dir = c(6,5))

    if (require(austin)) {
        wfmodelAustin <- wordfish(quanteda::as.wfm(ie2010dfm), dir = c(6,5))
        cor(wfmodel@theta, wfmodelAustin$theta)
    }
}
```

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, ...)

## 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	the fitted wordscores textmodel on which prediction will be made
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

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

Ken 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")
ws
wsp <- predict(ws)
wsp

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

texts *get or set corpus texts*

Description

Get or replace the texts in a quanteda corpus object.

Usage

```
texts(corp)

texts(corp) <- value
```

Arguments

`corp`	A quanteda corpus object
`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.

Examples

```
texts(inaugCorpus)[1]
sapply(texts(inaugCorpus), nchar)  # length in characters of the inaugual corpus texts

## this doesn't work yet - need to overload `[` for this replacement function
# texts(inaugTexts)[55] <- "GW Bush's second inaugural address, the condensed version."
```

tokenize *tokenize a set of texts*

Description

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

Usage

```
tokenize(x, ...)

## S3 method for class 'character'
tokenize(x, simplify = FALSE, sep = " ", ...)

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

Arguments

`x`	The text(s) or corpus to be tokenized
`...`	additional arguments passed to `clean`
`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.
`sep`	by default, tokenize expects a "white-space" delimiter between tokens. Alternatively, `sep` can be used to specify another character which delimits fields.

Value

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

Examples

```
# same for character vectors and for lists
tokensFromChar <- tokenize(inaugTexts)
tokensFromCorp <- tokenize(inaugCorpus)
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)

# demonstrate some options with clean
head(tokenize(inaugTexts[57], simplify=TRUE, lower=FALSE), 30)
```

`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(inaugCorpus))
topfeatures(dfm(inaugCorpus, stopwords=TRUE))
# least frequent features
topfeatures(dfm(inaugCorpus), 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, nsample = NULL, verbose = TRUE)

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

trimdfm(x, ...)
```

Arguments

`x`	document-feature matrix of dfm-class
`minCount`	minimum feature count
`minDoc`	minimum number of documents in which a feature appears
`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

Author(s)

Ken Benoit, inspired by code by Will Lowe (see trim)

Examples

```
dtm <- dfm(inaugCorpus)
dim(dtm)
dtmReduced <- trim(dtm, minCount=10, minDoc=2) # only words occuring >=5 times and in >=2 docs
dim(dtmReduced)
topfeatures(dtmReduced, decreasing=FALSE)
dtmSampled <- trim(dtm, minCount=20, nsample=50)  # sample 50 words over 20 count
dtmSampled # 57 x 50 words
topfeatures(dtmSampled)
```

`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

```
data(ukimmigTexts)
ukimmigCorpus <- corpus(ukimmigTexts, docvars=list(party=names(ukimmigTexts)))
language(ukimmigCorpus) <- "english"
encoding(ukimmigCorpus) <- "UTF-8"
summary(ukimmigCorpus)
```

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

Description

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

Usage

```
weight(x, ...)

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

tf(x)

tfidf(x)
```

```
smoother(x, smooth)

weighting(object)

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

Arguments

`x`	document-feature matrix created by dfm
`...`	not currently used
`type`	The weighting function to aapply to the dfm. One of: • normTf - Length normalization: dividing the frequency of the feature by the length of the document) • logTf - The natural log of the term frequency • tf-idf - 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. • maxTf - The term frequency divided by the frequency of the most frequent term in the document • ppmi - Positive Pointwise Mutual Information
`smooth`	amount to apply as additive smoothing to the document-feature matrix prior to weighting, default is 0.5, set to `smooth=0` for no smoothing.
`normalize`	if `TRUE` (default) then normalize the dfm by relative term frequency prior to computing tfidf
`verbose`	if `TRUE` output status messages
`object`	the dfm object for accessing the weighting setting

Details

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

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

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

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

Value

The dfm with weighted values

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

Author(s)

Paul Nulty and Kenneth Benoit

References

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

Examples

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

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

`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 Dr. Martin Porter's stemming algorithm and the C libstemmer library generated by Snowball.

Usage

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

Arguments

`words`	a character vector of words whose stems are to be extracted.
`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.loc.gov/standards/iso639-2/php/code_list.php` for a list of ISO-639 language codes

See Also

wordStem

Examples

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

Index