R text analysis: quanteda

Kasper Welbers & Wouter van Atteveldt

2018-09

This tutorial

In this tutorial you will learn how to perform text analysis using the quanteda package. In the R Basics: getting started tutorial you were already given code to create wordclouds, compare term frequenties in two corpora and create keyword-in-context listings. This time, the goal is to understand this code better, towards becoming more self-proficient in using quanteda.

The quanteda package

The quanteda package is an extensive text analysis suite for R. It covers everything you need to perform a variety of automatic text analysis techniques, and features clear and extensive documentation. Here we'll focus on the main preparatory steps for text analysis, and on learning how to browse the quanteda documentation. The documentation for each function can also be found here.

For a more detailed explanation of the steps discussed here, you can read the paper Text Analysis in R (Welbers, van Atteveldt & Benoit, 2017).

```
library(quanteda)
```

Importing text and creating a quanteda corpus

The first step is getting text into R in a proper format. stored in a variety of formats, from plain text and CSV files to HTML and PDF, and with different 'encodings'. There are various packages for reading these file formats, and there is also the convenient readtext that is specialized for reading texts from a variety of formats.

For this tutorial, we will be importing text from a csv. For convenience, we're using a csv that's available online, but the process is the same for a csv file on your own computer. The data consists of the State of the Union speeches of US presidents, with each document (i.e. row in the csv) being a paragraph. The data will be imported as a data.frame.

```
url = 'https://bit.ly/2QoqUQS'
d = read.csv(url)
head(d)   ## view first 6 rows
```

We can now create a quanteda corpus with the `corpus()` function. If you want to learn more about this function, recall that you can use the question mark to look at the documentation.

```
?corpus
```

Here you see that for a data.frame, we need to specify which column contains the text field. Also, the text column must be a character vector. In data is imported from a csv, a `character` vector is sometimes interpreted as a `factor` vector. We'll therefore first explicitly convert the column to a `character` vector.

```
d$text = as.character(d$text)          ## force 'text' column to be a character vector
corp = corpus(d, text_field = 'text')  ## create the corpus
corp
```

Creating the DTM (or DFM)

Many text analysis techniques only use the frequencies of words in documents. This is also called the bag-of-words assumption, because texts are then treated as bags of individual words. Despite ignoring much relevant information in the order of words and syntax, this approach has proven to be very powerfull and efficient.

The standard format for representing a bag-of-words is as a `document-term matrix` (DTM). This is a matrix in which rows are documents, columns are terms, and cells indicate how often each term occured in each document. We'll first create a small example DTM from a few lines of text. Here we use quanteda's `dfm()` function, which stands for `document-feature matrix` (DFM), which is a more general form of a DTM.

```
text <-  c(d1 = "Guns are awesome!",
           d2 = "We need better gun control!",
           d3 = "This is a sentence about cheese.")

dtm <- dfm(text, tolower=F)
dtm
```

Here you see, for instance, that the word `cheese` only occurs in the third document. In this matrix format, we can perform calculations with texts, like analyzing different sentiments of frames regarding guns, or the fact that the third sentence has nothing to do with the first two sentences.

However, directly converting a text to a DTM is a bit crude. Note, for instance, that the words `Gun` and `guns` are given different columns. In this DTM, "Gun" and "guns" are as different as "gun" and "cheese", but for many types of analysis we would be more interested in the fact that both texts are about guns, and not about the specific word that is used. Also, for performance it can be usefull (or necessary) to use fewer columns, and to ignore less interesting words such as `is`.

This can be achieved by using additional `preprocessing` steps. In the next example, we'll again create the DTM, but this time we make all text lowercase, ignore stopwords and punctuation, and perform `stemming`. Simply put, stemming removes some parts at the ends of words to ignore different forms of the same word, such as singular versus plural ("gun" or "gun-s") and different verb forms ("walk","walk-ing","walk-s")

```
dtm = dfm(text, tolower=T, remove = stopwords('en'), stem = T, remove_punct=T)
dtm
```

By now you should be able to understand better how the arguments in this function work. The `tolower` argument determines whether texts are (TRUE) or aren't (FALSE) converted to lowercase. `stem` determines whether stemming is (TRUE) or isn't (FALSE) used. The remove argument is a bit more tricky. If you look at the documentation for the dfm function (`?dfm`) you'll see that `remove` can be used to give "a pattern of user-supplied features to ignore". In this case, we actually used another function, `stopwords()`, to get a list of english stopwords. You can see for yourself.

```
stopwords('en')
```

This list of words is thus passed to the `remove` argument in the `dfm()` to ignore these words. If you are using texts in another language, make sure to specify the language, such as stopwords('nl') for Dutch or stopwords('de') for German.

There are various alternative preprocessing techniques, including more advanced techniques that are not implemented in quanteda. Whether, when and how to use these techniques is a broad topic that we won't cover today. For more details about preprocessing you can read the Text Analysis in R paper.

Filtering the DTM

For this tutorial, we'll use the State of the Union speeches. We already created the corpus above. We can now pass this corpus to the `dfm()` function and set the preprocessing parameters.

```
dtm = dfm(corp, tolower=T, stem=T, remove=stopwords('en'), remove_punct=T)
dtm
```

This dtm has 23,469 documents and 20,469 features (i.e. terms), and no longer shows the actual matrix because it simply wouldn't fit.

A final step in our data preparation is to filter out some of less important terms. For didactic reasons, we'll walk through the steps. In the previous tutorial you learned that selection works by comparing a vector to a given value, which then selects all elements for which the comparison is TRUE. To filter on the frequency of terms, we can get the column sums of the matrix (the sum of all values in a column) and see whether the value is higher than a given threshold. Here we use threshold 10.

```
## we'll use head() in the following examples to only show
## the first 6 values (instead of 20,469)
head(colSums(dtm))      ## sums for first six columns/terms
head(colSums(dtm) > 10) ## terms for which value is higher than 20
```

We can use this comparison in the `dfm_select()` function to select the columns for which the comparison is TRUE (sum is higher than 20).

```
dtm = dtm[, colSums(dtm) > 10] ## read: select columns in dtm where column sum > 10
dtm
```

Now we have about 5000 features left.

Analysis

Using the dtm we can now employ various techniques. You've already seen some of them in the first tutorial, but by now you should be able to understand more about the R syntax, and understand how to tinker with different parameters.

Word frequencies and wordclouds

Get most frequent words in corpus.

```
textplot_wordcloud(dtm, max_words = 50)     ## top 50 (most frequent) words
```

```
textplot_wordcloud(dtm, max_words = 50, color = c('blue','red')) ## change colors
```

```
textstat_frequency(dtm, n = 10)             ## view the frequencies
```

You can also inspect a subcorpus. For example, looking only at Obama speeches. To subset the DTM we can use quanteda's `dtm_subset()`, but we can also use the more general R subsetting techniques (as discussed last week). Here we'll use the latter for illustration.

With `docvars(dtm)` we get a data.frame with the document variables. With `docvars(dtm)$President`, we get the character vector with president names. Thus, with `docvars(dtm)$President == 'Barack Obama'` we look for all documents where the president was Obama. To make this more explicit, we store the logical vector, that shows which documents are 'TRUE', as is_obama. We then use this to select these rows from the DTM.

```
is_obama = docvars(dtm)$President == 'Barack Obama'
obama_dtm = dtm[is_obama,]
textplot_wordcloud(obama_dtm, max_words = 25)
```

Compare corpora

Compare word frequencies between two subcorpora. Here we (again) first use a comparison to get the is_obama vector. We then use this in the `textstat_keyness()` function to indicate that we want to compare the Obama documents (where is_obama is TRUE) to all other documents (where is_obama is FALSE).

```
is_obama = docvars(dtm)$President == 'Barack Obama'
ts = textstat_keyness(dtm, is_obama)
head(ts, 20)    ## view first 20 results
```

We can visualize these results, stored under the name `ts`, by using the textplot_keyness function

```
textplot_keyness(ts)
```

Keyword-in-context

As seen in the first tutorial, a keyword-in-context listing shows a given keyword in the context of its use. This is a good help for interpreting words from a wordcloud or keyness plot.

Since a DTM only knows word frequencies, the `kwic()` function requires the corpus object as input.

```
k = kwic(corp, 'freedom', window = 7)
head(k, 10)    ## only view first 10 results
```

The `kwic()` function can also be used to focus an analysis on a specific search term. You can use the output of the kwic function to create a new DTM, in which only the words within the shown window are included in the DTM. With the following code, a DTM is created that only contains words that occur within 10 words from `terror*` (terrorism, terrorist, terror, etc.).

```
terror = kwic(corp, 'terror*')
terror_corp = corpus(terror)
terror_dtm = dfm(terror_corp, tolower=T, remove=stopwords('en'), stem=T, remove_punct=T)
```

Now you can focus an analysis on whether and how Presidents talk about `terror*`.

```
textplot_wordcloud(terror_dtm, max_words = 50)     ## top 50 (most frequent) words
```

Dictionary search

You can perform a basic dictionary search. In terms of query options this is less advanced than AmCAT, but quanteda offers more ways to analyse the dictionary results. Also, it supports the use of existing dictionaries, for instance for sentiment analysis (but mostly for english dictionaries).

An convenient way of using dictionaries is to make a DTM with the columns representing dictionary terms.

```
dict = dictionary(list(terrorism = 'terror*',
                       economy = c('econom*', 'tax*', 'job*'),
                       military = c('army','navy','military','airforce','soldier'),
                       freedom = c('freedom','liberty')))
dict_dtm = dfm(corp, dictionary = dict)
dict_dtm
```

The "4 features" are the four entries in our dictionary. Now you can perform all the analyses with dictionaries.

```
textplot_wordcloud(dict_dtm)
```

```
tk = textstat_keyness(dict_dtm, docvars(dict_dtm)$President == 'Barack Obama')
textplot_keyness(tk)
```

Topic modeling

Topic modeling is a method for automatically findings topics in a corpus. These topics are then respresented as clusters of words that are indicative of the topic (also see Text Analysis in R or Jacobi et al. for more details). It has often been used as a tool for exploring huge corpora. For smaller corpora (and the current corpus is in that sense small to moderate size) it is often less usefull.

Topic modeling is not directly supported in quanteda, but quanteda is designed to easily prepare data for use in external packages such as the `topicmodels` package. To illustrate this, we'll use the topicmodels package here to create a topicmodel. If you want to perform these steps, you'll first have to install the topicmodels package (only the first time).

```
install.packages('topicmodels')
```

```
library(topicmodels)

topmod_dtm <- convert(dtm, to = "topicmodels")

## create a topicmodel with 5 topics and some additional settings
lda <- LDA(topmod_dtm, k = 10, method = 'Gibbs')
terms(lda, 10)
```

Scrabble word value

Not very usefull, but yeah.

```
nscrabble('nexus')
```