TEXT AS DATA: WEEK 6

MATTHIAS HABER

13 OCTOBER 2021

GOALS FOR TODAY

GOALS

- Quanteda
- Text preprocessing
- Document-Feature Matrix

QUANTEDA

QUANTEDA

The *quanteda* package is an extensive text analysis suite for R, containing everything you need to perform a variety of automatic text analyses. *quanteda* is to text analysis what dplyr and tidyr are to data wrangling.

quanteda is built for efficiency and speed, through its design around three infrastructures: the stringi package for text processing, the Matrix package for sparse matrix objects, and computationally intensive processing (e.g. for tokens) handled in parallelized C++.

In addition to the extensive documentation, there's a very helpful cheatsheet here

```
library(quanteda) #install.packages("quanteda")
```

BUILD-IN CORPUSES IN QUANTEDA

`quanteda` comes with several corpora included, like the corpus of US presidential inaugural addresses:

```
corp <- quanteda::data_corpus_inaugural
summary(corp)
```

```
## Corpus consisting of 59 documents, showing 59 documents:
##
##             Text Types Tokens Sentences Year  President       FirstNa
##  1789-Washington   625   1537        23 1789 Washington          Geor
##  1793-Washington    96    147         4 1793 Washington          Geor
##       1797-Adams   826   2577        37 1797      Adams            Jo
##   1801-Jefferson   717   1923        41 1801  Jefferson          Thom
##   1805-Jefferson   804   2380        45 1805  Jefferson          Thom
##     1809-Madison   535   1261        21 1809    Madison           Jam
##     1813-Madison   541   1302        33 1813    Madison           Jam
##      1817-Monroe  1040   3677       121 1817     Monroe           Jam
##      1821-Monroe  1259   4886       131 1821     Monroe           Jam
##      1825-Adams   1003   3147        74 1825      Adams     John Quin
##     1829-Jackson   517   1208        25 1829    Jackson          Andr
##     1833-Jackson   499   1267        29 1833    Jackson          Andr
##    1837-VanBuren  1315   4158        95 1837  Van Buren          Mart
##    1841-Harrison  1898   9123       210 1841   Harrison   William Hen
##        1845-Polk  1334   5186       153 1845       Polk       James K
```

PREPROCESSING

PREPROCESSING STEPS

Preprocessing text for analysis typically involves 8 steps:

1. Tokenizing the text to unigrams (or bigrams, trigrams, sentences, etc.)
2. Converting all characters to lowercase
3. Removing punctuation
4. Removing numbers & symbols
5. Removing stop words, including custom stop words
6. “Stemming” words, or lemmatization
7. Creating a Document-Feature Matrix
8. Removing sparse or frequent terms

TOKENIZATION

The `token()` function from *quanteda* allows to use external or the internal tokenizer to construct a token object, By default it creates word tokens and preserves hyphens, URLs, social media "tag" characters, and email addresses.

```
txt <- c(text1 = "This is a short text,\n separated across two lines.",
         text2 = "Text-As-Data:
        https://github.com/mhaber/Text_as_Data_21")
tokens(txt)
```

```
## Tokens consisting of 2 documents.
## text1 :
##  [1] "This"      "is"        "a"         "short"     "text"      ","
##  [7] "separated" "across"    "two"       "lines"     "."
##
## text2 :
## [1] "Text-As-Data"
## [2] ":"
## [3] "https://github.com/mhaber/Text_as_Data_21"
```

TOKENIZATION

You can use the `what` argument to change the type of token object you want to create. For example, if you want it to construct sentences:

```
tokens(c("Kurt Vonnegut said; only assholes use semi-colons.",
         "To be? Or not to be?"), what = "sentence")
```

```
## Tokens consisting of 2 documents.
## text1 :
## [1] "Kurt Vonnegut said; only assholes use semi-colons."
##
## text2 :
## [1] "To be?"         "Or not to be?"
```

TOKENIZATION

or characters:

```
tokens("Just a bunch of characters.", what = "character")
```

```
## Tokens consisting of 1 document.
## text1 :
##  [1] "J" "u" "s" "t" "a" "b" "u" "n" "c" "h" "o" "f"
## [ ... and 11 more ]
```

TOKENIZATION

If you want to create n-grams in any length you can use the `tokens_ngrams()` function on a tokenized object.

```
toks <- tokens(data_char_ukimmig2010[[3]])
tokens_ngrams(toks, n = 2:4) %>% head(10)
```

```
## Tokens consisting of 1 document.
## text1 :
##  [1] "Attract_the"     "the_brightest"   "brightest_and"   "and_best"
##  [5] "best_to"         "to_our"          "our_country"     "country_.
##  [9] "._Immigration"   "Immigration_has" "has_enriched"    "enriched_
## [ ... and 1,479 more ]
```

TOKENIZATION

You can also use other tokenizers like the ones contained in the `tokenizers` library.

```
library(tokenizers)
song <-  paste0("How many roads must a man walk down\n",
                "Before you call him a man?\n",
                "How many seas must a white dove sail\n",
                "Before she sleeps in the sand?\n",
                "\n",
                "How many times must the cannonballs fly\n",
                "Before they're forever banned?\n",
                "The answer, my friend, is blowin' in the wind.\n",
                "The answer is blowin' in the wind.\n")
tokenizers::tokenize_paragraphs(song)
```

```
## [[1]]
## [1] "How many roads must a man walk down Before you call him a man? Ho
## [2] "How many times must the cannonballs fly Before they're forever ba
```

EXERCISE

Tokenize the following tweet using quanteda's built-in word tokenizer, the tokenize_words tokenizer from the tokenizers package, and the tokenize_tweets tokenizer from the tokenizers package. What's the difference?

https://t.co/8z2f3P3sUc @datageneral FB needs to hurry up and add a laugh/cry button 😬😭😓🤢🙄😱 Since eating my feelings has not fixed the world's problems, I guess I'll try to sleep...

CONVERSION TO LOWER CASE

In text analysis, it usually makes sense to convert characters to lower case although sometimes you may want to keep proper names capitalized. In `quanteda` you can use the `char_tolower()` directly on strings:

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

```
##                                                text1
## "england and france are members of nato and unesco"
##                                                text2
##                   "nasa sent a rocket into space."
```

```
char_tolower(test1, keep_acronyms = TRUE)
```

```
##                                                text1
## "england and france are members of NATO and UNESCO"
##                                                text2
##                   "NASA sent a rocket into space."
```

CONVERSION TO LOWER CASE

or the `tokens_tolower()` function after you created your tokens object:

```
test2 <- tokens(test1)
tokens_tolower(test2, keep_acronyms = TRUE)
```

```
## Tokens consisting of 2 documents.
## text1 :
## [1] "england" "and"     "france"  "are"     "members" "of"      "NATO"
## [8] "and"     "UNESCO"
##
## text2 :
## [1] "NASA"   "sent"   "a"      "rocket" "into"   "space"  "."
```

REMOVING PUNCTUATION, NUMBERS & SYMBOLS

Removing unwanted features like punctuation, symbols and numbers is really easy and be done directly insight the `tokens` function:

```
text <- "ph4t, phat, ph@    is 1337 for awesome or cool"
tokens(text,
       remove_punct = TRUE,
       remove_numbers = TRUE,
       remove_symbols = TRUE)
```

```
## Tokens consisting of 1 document.
## text1 :
## [1] "ph4t"    "phat"    "ph"      "is"      "for"     "awesome" "or"
## [8] "cool"
```

REMOVING STOP WORDS

As we've seen in the previous sessions, we often may want to remove stop words from our tokens object. `quanteda` automatically loads the `stopwords()` function from the `stopwords` package and defaults to the Snowball collection. Use `stopwords_getsources()` to get a list of available sources and `stopwords_getlanguages` to get a list of available languages for each source.

```
stopwords::stopwords_getsources()
```

```
## [1] "snowball"      "stopwords-iso" "misc"          "smart"
## [5] "marimo"        "ancient"       "nltk"          "perseus"
```

```
stopwords::stopwords_getlanguages("snowball")
```

```
##  [1] "da" "de" "en" "es" "fi" "fr" "hu" "ir" "it" "nl" "no" "pt" "ro"
```

REMOVING STOP WORDS

We can remove stopwords from our tokens object with the `tokens_remove()` function.

```
text_token <- tokens("The quick brown fox jumps over the lazy dog")
tokens_remove(text_token, stopwords("en"))
```

```
## Tokens consisting of 1 document.
## text1 :
## [1] "quick" "brown" "fox"   "jumps" "lazy"  "dog"
```

REMOVING OTHER UNWANTED WORDS

We can also use `tokens_remove()` to remove other unwanted words from our tokens object:

```
tokens_remove(text_token, c("fox", "dog"))
```

```
## Tokens consisting of 1 document.
## text1 :
## [1] "The"   "quick" "brown" "jumps" "over"  "the"   "lazy"
```

EXERCISE

Remove all the stopwords from the following Turkish poem:

“Yaşamak bir ağaç gibi tek ve hür ve bir orman gibi kardeşçesine, bu hasret bizim.”

STEMMING

Stemming is the truncation of words to their root form, e.g. *playing*, *plays*, *played* become *play*.

The tokenizers package provides a wrapper to the wordStem function from the SnowballC package, which applies a standard stemmer called the Porter stemmer. The stemmer is available for the following languages:

```
SnowballC::getStemLanguages()
```

```
##  [1] "arabic"     "basque"     "catalan"    "danish"     "dutch"
##  [6] "english"    "finnish"    "french"     "german"     "greek"
## [11] "hindi"      "hungarian"  "indonesian" "irish"      "italian"
## [16] "lithuanian" "nepali"     "norwegian"  "porter"     "portuguese"
## [21] "romanian"   "russian"    "spanish"    "swedish"    "tamil"
## [26] "turkish"
```

STEMMING

To reduce words to their word stem we can use the `tokens_wordstem()` function and apply it again to a tokens object:

```
txt <- c(one = "eating eater eaters eats ate",
         two = "taxing taxes taxed my tax return")
th <- tokens(txt)
tokens_wordstem(th)
```

```
## Tokens consisting of 2 documents.
## one :
## [1] "eat"   "eater" "eater" "eat"   "ate"
##
## two :
## [1] "tax"    "tax"    "tax"    "my"     "tax"    "return"
```

ALL THE PREPROCESSING IN ONE GO

```
# load data corpus
corp <- quanteda::data_corpus_inaugural
#create token
inaugural_tokens <- tokens(corp,
                      what = "word",
                      remove_punct = TRUE,
                      remove_symbols = TRUE,
                      remove_numbers = FALSE
                      )
# apply further preprocessing
inaugural_tokens_reduced <- inaugural_tokens %>%
  tokens_tolower() %>%
  tokens_remove(stopwords('en')) %>%
  tokens_wordstem()
```

KEYWORDS IN CONTEXT

A useful feature built into `quanteda` is keywords in context, which returns all the appearances of a word (or combination of words) in its immediate context.

```
kwic(inaugural_tokens, "humble", window=3)
```

```
## Keyword-in-context with 13 matches.
##   [1789-Washington, 572]          along with an | humble |
##  [1789-Washington, 1359]          Human Race in | humble |
##       [1797-Adams, 2123]           age and with | humble |
##    [1801-Jefferson, 169] the contemplation and | humble |
##       [1821-Monroe, 173]             favor of my | humble |
##       [1825-Adams, 2902]          I commit with | humble |
##       [1829-Jackson, 85]       dedication of my | humble |
##       [1833-Jackson, 91]           extent of my | humble |
##      [1853-Pierce, 3174]        in the nation's | humble |
##    [1857-Buchanan, 1204]                I feel an | humble |
##   [1953-Eisenhower, 765]             of the most | humble |
##      [1997-Clinton, 586]          a new century | humble |
##       [2009-Obama, 1760]       we remember with | humble |
##
##  anticipation of the
##  supplication that since
```

reverence | feel

EXERCISE

1. How many times has the word “slave” been used in inaugural addresses?
2. How many times has a word that included “slave” (like “slavery” or “enslaved”) been used in inaugural addresses?

DOCUMENT-FEATURE-MATRIX

QUANTEDA DFM

Quanteda is focused on bag-of-words (or bag-of-tokens) models that work from a document-feature matrix, where each row represents a document, each column represents a type (a "term" in the vocabulary) and the entries are the counts of tokens matching the term in the current document.

To create a document-feature maxtrix use the `dfm()` function and apply it directly to the tokens object along with some common preprocessing options:

```
my_dfm <- quanteda::dfm(inaugural_tokens,
                        tolower = TRUE)
```

DFM SUMMARY

Typing the dfm's name will show an object summary.

```
my_dfm
```

```
## Document-feature matrix of: 59 documents, 9,422 features (91.89% spars
##                  features
## docs              fellow-citizens  of the senate and house representat
##   1789-Washington               1  71 116      1  48     2
##   1793-Washington               0  11  13      0   2     0
##   1797-Adams                    3 140 163      1 130     0
##   1801-Jefferson                2 104 130      0  81     0
##   1805-Jefferson                0 101 143      0  93     0
##   1809-Madison                  1  69 104      0  43     0
##                  features
## docs              among vicissitudes incident
##   1789-Washington     1            1        1
##   1793-Washington     0            0        0
##   1797-Adams          4            0        0
##   1801-Jefferson      1            0        0
##   1805-Jefferson      7            0        0
##   1809-Madison        0            0        0
## [ reached max ndoc ... 53 more documents, reached max nfeat ... 9,412
```

DFM SUBSETTING

You can look inside your dfm by indexing it like you would a Matrix object:

```
my_dfm[1:5,1:5]
```

```
## Document-feature matrix of: 5 documents, 5 features (20.00% sparse) a
##                  features
## docs              fellow-citizens  of the senate and
##   1789-Washington                1  71 116      1  48
##   1793-Washington                0  11  13      0   2
##   1797-Adams                     3 140 163      1 130
##   1801-Jefferson                 2 104 130      0  81
##   1805-Jefferson                 0 101 143      0  93
```

DFM TOPFEATURES

You can list the most (or least) frequently occuring features in a dfm by using the `topfeatures()` function:

```
topfeatures(my_dfm, 40)
```

```
##        the         of        and         to         in          a 
##      10183       7180       5406       4591       2827       2292 
##         we       that         be         is         it        for 
##       1827       1813       1502       1491       1398       1230 
##       have      which        not       with         as       will 
##       1031       1007        980        970        966        944 
##          i        all        are      their        but        has 
##        871        836        828        761        670        631 
##       from        its government         or         on         my 
##        578        573        564        563        544        515 
##       been        can         no       they         so           
##        496        487        470        463        397
```

DFM_TRIM()

You can also reduce the size of your dfm by removing sparse or frequently appearing terms with the `dfm_trim()` function. For example, to keep only those words that occur at least 10 times but not more than 100 times:

```
dfm_trim(my_dfm, min_termfreq = 10, max_termfreq = 100)
```

```
## Document-feature matrix of: 59 documents, 1,381 features (74.22% spars
##                  features
## docs              fellow-citizens senate house representatives event c
##   1789-Washington               1      1     2                2     2
##   1793-Washington               0      0     0                0     0
##   1797-Adams                    3      1     0                2     0
##   1801-Jefferson                2      0     0                0     0
##   1805-Jefferson                0      0     0                0     0
##   1809-Madison                  1      0     0                0     0
##                  features
## docs              greater order received day
##   1789-Washington       1     2        1   2
##   1793-Washington       0     0        0   0
##   1797-Adams            0     4        0   1
##   1801-Jefferson        1     1        0   1
```

```
##   1805-Jefferson          0     3         0    1
##   1809-Madison            0     0         0    0
```

EXERCISE

Construct a `dfm` of trigrams from the inaugural speeches (lower case, no punctuation, not stemmed, no stop words removed).

1. How big is the matrix? How sparse is it?
2. What are the 20 most frequent trigrams?
3. Keep only those trigrams occurring at least 50 times and in at least 3/4 of the documents

WRAPPING UP

QUESTIONS?

OUTLOOK FOR OUR NEXT SESSION

- Next week there is no class!
- We'll meet again on October 27 where we'll learn different ways to visualize text
- I'll upload the code completion exercise this week. Please hand in your solution as an `.Rmd` file by October 26.

THAT'S IT FOR TODAY

Thanks for your attention!