Text Analysis with R

Claudia Engel, Scott Bailey

Last updated: April 28, 2019

Contents

Prerequisites

- You should have some **basic knowledge** of R, and be familiar with the topics covered in the Introduction to R.
- Have a recent version of R and RStudio installed.
- Libraries:
  - `tidyverse`
  - `tidytext`
  - `readtext`
  - `sotu`
  - `tm`
  - `quanteda`

References

Feinerer, I., Hornik, K., and Meyer, D. (2008). Text Mining Infrastructure in R. Journal of Statistical Software, 25(5), 1 - 54. doi:http://dx.doi.org/10.18637/jss.v025.i05

Gries, Stefan Thomas, 2009: Quantitative Corpus Linguistics with R: A Practical Introduction. Routledge.

Silge, J and D. Robinson, 2017: Text Mining with R: A Tidy Approach

Kasper Welbers, Wouter Van Atteveldt & Kenneth Benoit (2017) Text Analysis in R, Communication Methods and Measures, 11:4, 245-265, DOI: 10.1080/19312458.2017.1387238

CRAN Task View: Natural Language Processing

Chapter 1

Analysing Texts

Learning Objectives

- to come

We'll use several libraries today. sotu will provide the metadata and text of State of the Union speeches ranging from George Washington to Barack Obama. tidyverse provides many of the standard "verbs" for working with our data. tidytext provides specific functions for a "tidy" approach to working with textual data. readtext provides a function well suited to reading textual data from a large number of formats into R.

```
library(sotu)
library(tidyverse)
library(tidytext)
library(readtext)
```

1.1 Reading text into R

First, let's look at the data in the sotu package. The metadata and texts come separately. We'll use the supplied metadata object, but we're going to use a utility function (sotu_dir) in the package to write the texts to disk so that we can practice reading text files from disk.

```
# Let's take a quick look at the state of the union metadata
summary(sotu_meta)
# sotu_dir writes the text files to a temporary dir, but you could specific where you want them.
fp <- sotu_dir()
head(fp)
```

Now that we have the files in disk, and a list of filepaths stored in the fp variable, we can use readtext to read the texts into a new variable.

```
# let's then read in the files with readtext
texts <- readtext(fp)
head(texts)
```

So that we can work with a single tabular dataset with a tidy approach, we'll convert the metadata and text tables to tibbles, and combine them into a single tibble. You can see that our texts are organized by alphabetical order, so first we'll need to sort our metadata to match.

```r
sotu_meta_tib <- as_tibble(sotu_meta) %>%
  arrange(president)

head(sotu_meta_tib)
```

```r
# We can now combine the sotu metadata with the texts
# first, we'll turn both pieces of data into tibbles, then combine
sotu_texts <- as_tibble(texts)
sotu_whole <- bind_cols(sotu_meta_tib, sotu_texts)
glimpse(sotu_whole)
```

Now that we have our data, we need to think about cleaning it. Depending on the quality of your data, you might need to explicitly replace certain characters or words, remove urls or types of numbers, such as phone numbers, or otherwise clean up misspellings or errors. There are several ways to handle this sort of cleaning, but we'll look at some straightforward string manipulation and replacement.

1.2 String operations

TODO

(For spell check: https://CRAN.R-project.org/package=spelling, https://CRAN.R-project.org/package=hunspell)

1.3 Tokenize, lowercase

A very common part of data cleaning involves tokenization. While our data is already "tidy" insofar as each row is a single observation, a single text with metdata, the tidytext approach goes a step further to make each word it's own observation with metadata. We could write our own function to do this using a tokenizer, but `tidytext` provides a handy utility function just for this purpose.

```r
tidy_sotu <- sotu_whole %>%
  unnest_tokens(word, text)

tidy_sotu
```

Before we move on, we should note that the `unnest_tokens` function didn't just tokenize our texts at the word level. It also lowercased each word, and it could do quite a bit more. For instance, we could tokenize the text at the level of ngrams or sentences, if those are the best units of analysis for our work. We could also leave punctuation, which has been removed by default. Depending on what you need to do for analysis, you use do these operations during this step, or write custom functions and do it before you unnest tokens.

1.4 Stopwords

Another common type of cleaning in text analysis is to remove stopwords, or common words that theoretically provide less information about the content of a text. Depending on the type of analysis you're doing, you might leave these words in or use a highly curated list of stopwords. For now, as we move toward looking at words in documents based on frequency, we will remove some standard stopwords using a tidytext approach.

First, let's look at the stopwords that tidytext gives us to get a sense of what they are.

```
data(stop_words)
head(stop_words, n = 60)
```

You can see that we now have one word per row with associated metadata. We can now remove stopwords using an `anti-join`.

```
tidy_sotu_words <- tidy_sotu %>%
  anti_join(stop_words)
```

```
#> Joining, by = "word"
```

```
tidy_sotu_words
```

We went from 1,965,212 to 778,161 rows, which means we had a lot of stopwords in our corpus. This is a huge removal, so for serious analysis, we might want to take a closer look at the stopwords and determine if we should use a different stopword list or otherwise create our own.

```
# Word tokenization with punctuation
tidy_sotu_w_punct <- sotu_whole %>%
  unnest_tokens(word, text, strip_punct = FALSE)

tidy_sotu_w_punct

# Sentence tokenization
tidy_sotu_sentences <- sotu_whole %>%
  unnest_tokens(sentence, text, token = "sentences", to_lower = FALSE)

tidy_sotu_sentences

# N-gram tokenization
tidy_sotu_trigram <- sotu_whole %>%
  unnest_tokens(trigram, text, token = "ngrams", n = 3)

tidy_sotu_trigram
```

1.5 Stemming and Lemmatization?

- hunspell, SnowballC?

1.6 Tagging

TODO?: Tag text with cleanNLP maybe?

Now that we've read in our text and metadata, reshaped it a bit into the tidytext format, and cleaned it a bit while doing so, let's move on to some basic analysis.

Chapter 2

Preparing Textual Data

Learning Objectives

- to come

- DTM
- n-grams
- co-ocurrence

First, we'll load the libraries we need.

```
library(tidyverse)
library(tidytext)
```

Let's remind ourselves of what our data looks like.

```
tidy_sotu_words
```

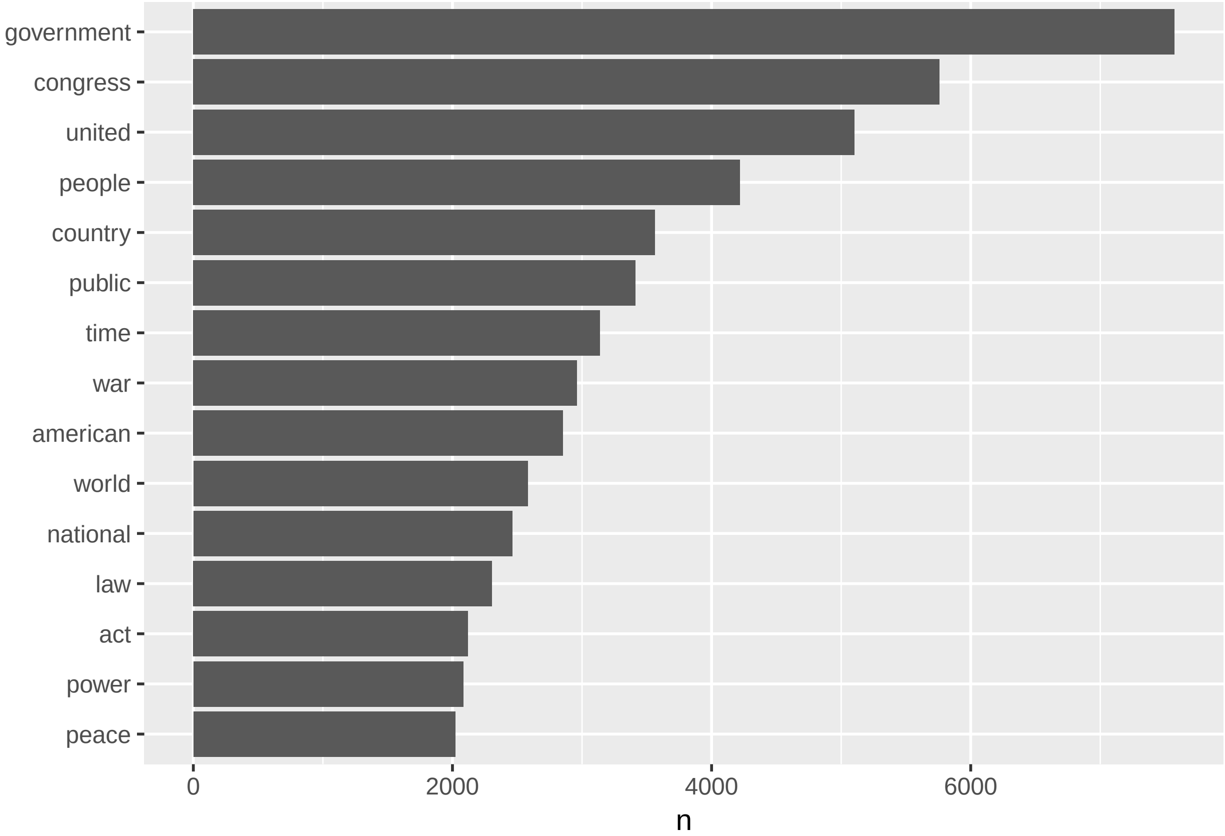
2.1 Frequencies

Since our unit of analysis at this point is a word, let's do some straightforward counting to figure out which words occur most frequently in the corpus as a whole.

```
tidy_sotu_words %>%
  count(word, sort = TRUE)
```

We could start adding in a bit of visualization here. Let's show the most frequent words that occur more than 2000 times.

```
tidy_sotu_words %>%
  count(word, sort = TRUE) %>%
  filter(n > 2000) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n)) +
  geom_col() +
  xlab(NULL) +
  coord_flip()
```



What if we're interested in most used words per speech?

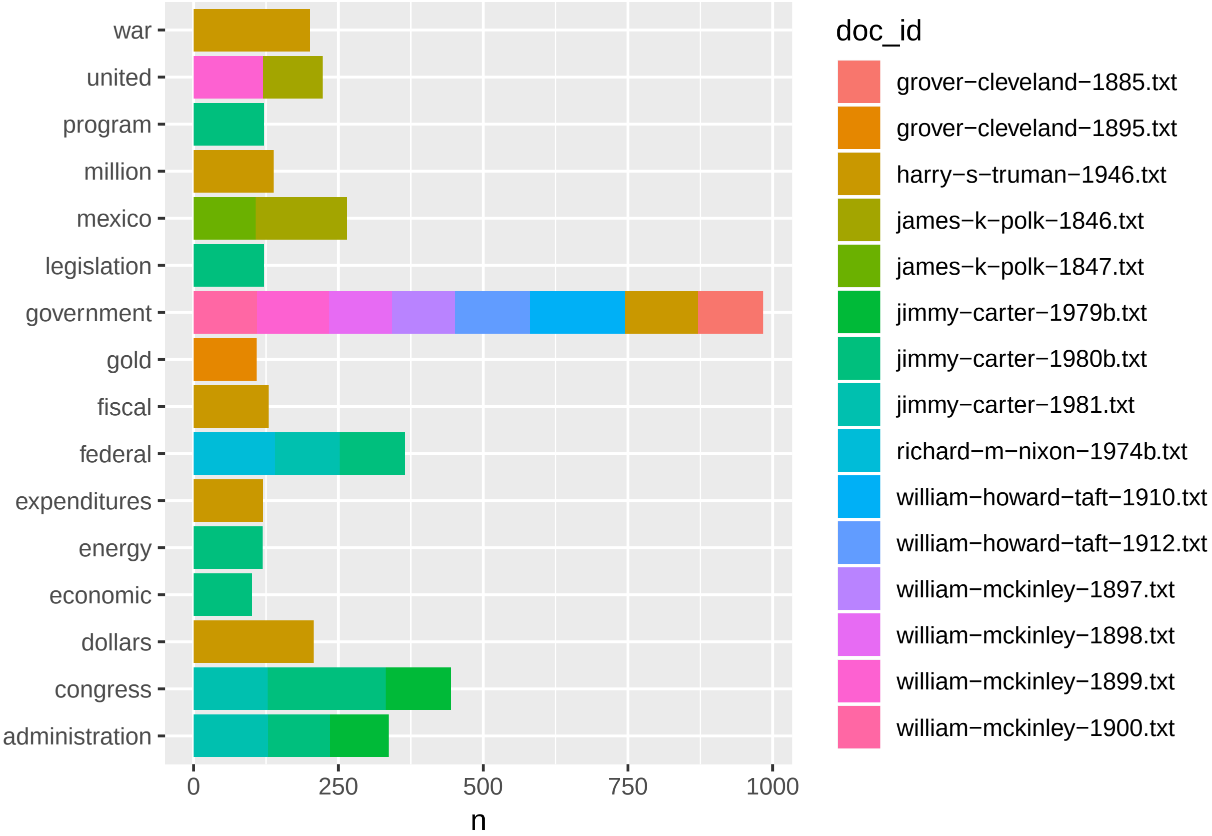
```
# Count words by book
doc_words <- tidy_sotu_words %>%
  count(doc_id, word, sort = TRUE)

# Calculate the total number of words by book and save them to a tibble
total_words <- doc_words %>%
  group_by(doc_id) %>%
  summarize(total = sum(n))

# Join the total column with the rest of the data so we can calculate frequency
doc_words <- left_join(doc_words, total_words)
```

```
#> Joining, by = "doc_id"
```

```
doc_words
```

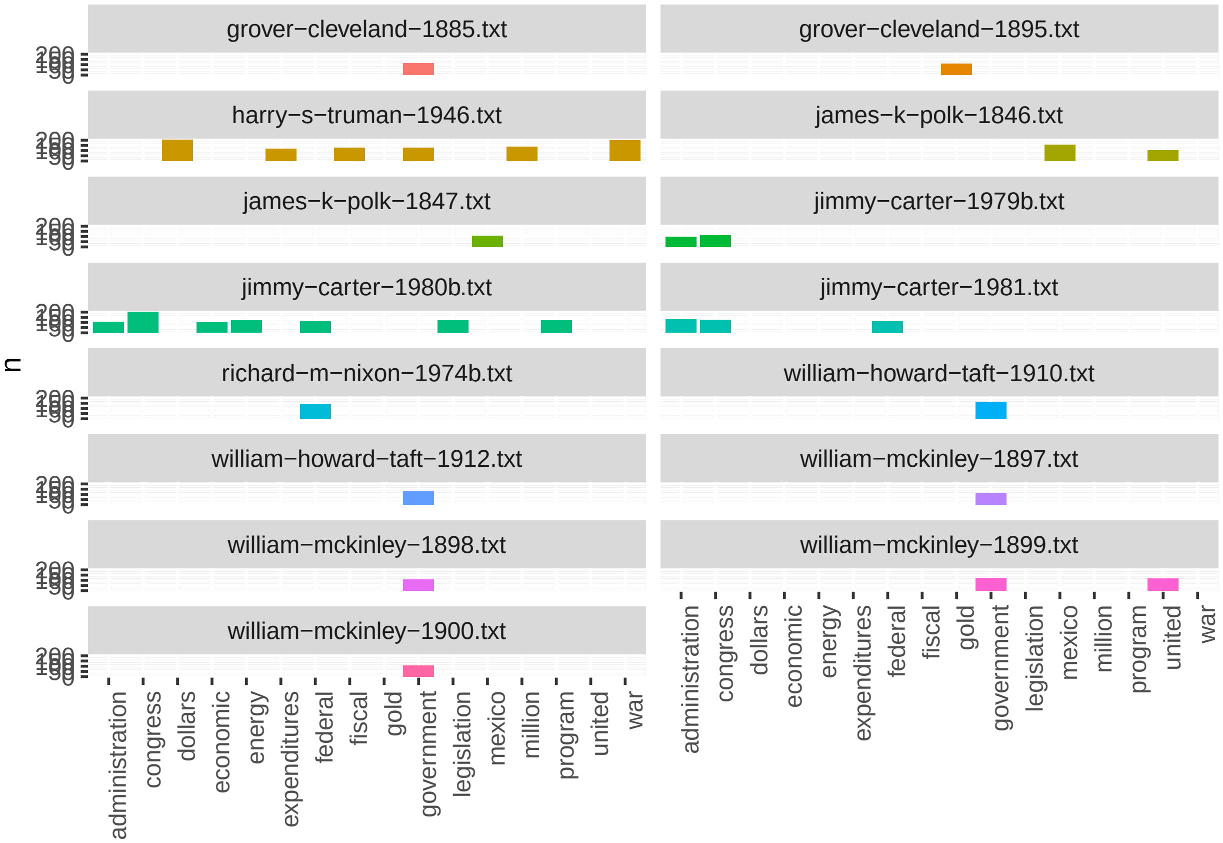
Let's graph the top words per book

```
doc_words %>%
  filter(n > 100) %>%
  ggplot(aes(word, n, fill = doc_id)) +
  geom_col() +
  xlab(NULL) +
  coord_flip()
```



That's cool looking, but let's split it into facets so we can see by speech.

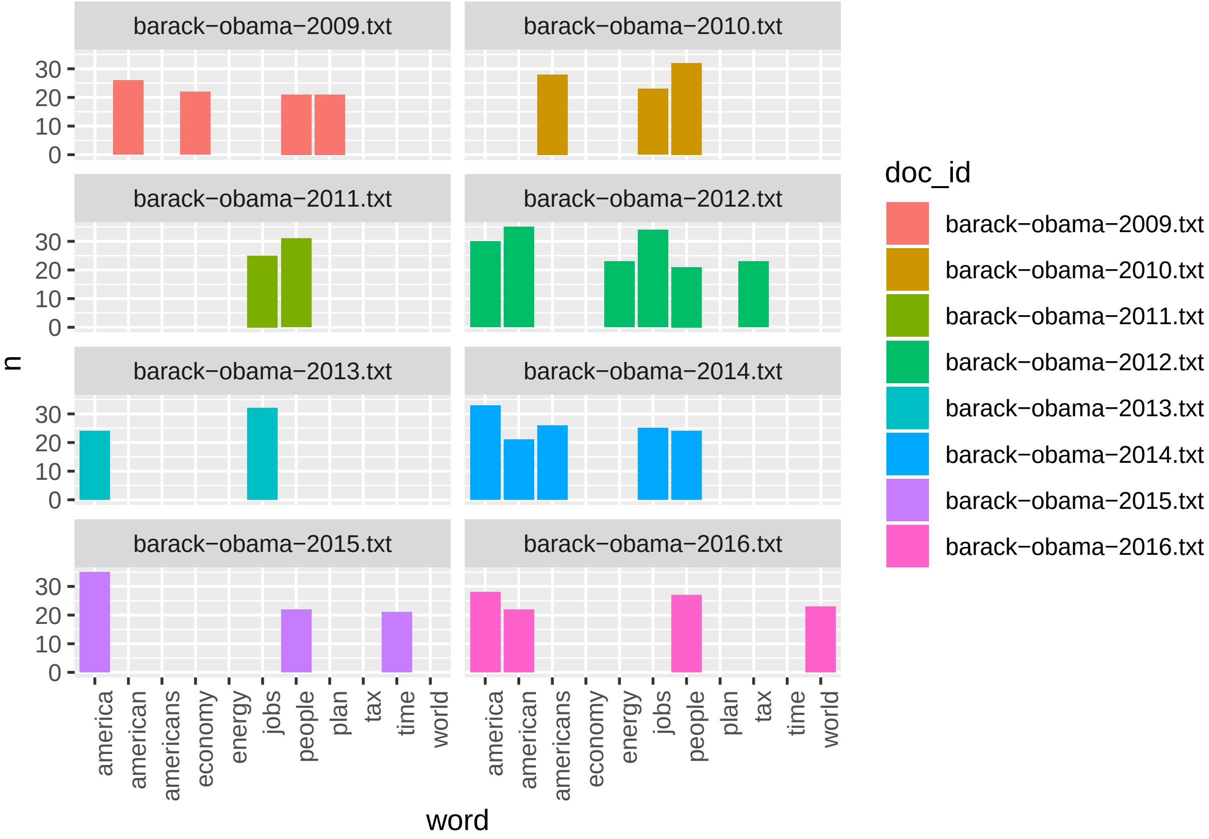
```
doc_words %>%
  filter(n > 100) %>%
  ggplot(aes(word, n, fill = doc_id)) +
  geom_col(show.legend = FALSE) +
  xlab(NULL) +
  facet_wrap(~doc_id, ncol = 2) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



We could keep cleaning this figure up by setting some minimum sizing, determining the spacing between y-axis labels better, and so forth, but now we'll accept it as showing some sense of variation across speeches where certain words are used most.

What if we want to check the most highly common words per speech for a single president? We could filter this `doc_words` dataset based on the president's name being in the doc_id, but I think it's easier to filter from the initial tidy data and recount.

```r
tidy_sotu_words %>%
  filter(president == "Barack Obama") %>%
  count(doc_id, word, sort = TRUE) %>%
  filter(n > 20) %>%
  ggplot(aes(word, n, fill=doc_id)) +
  geom_col() +
  facet_wrap(~doc_id, ncol = 2) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



2.2 Term frequency

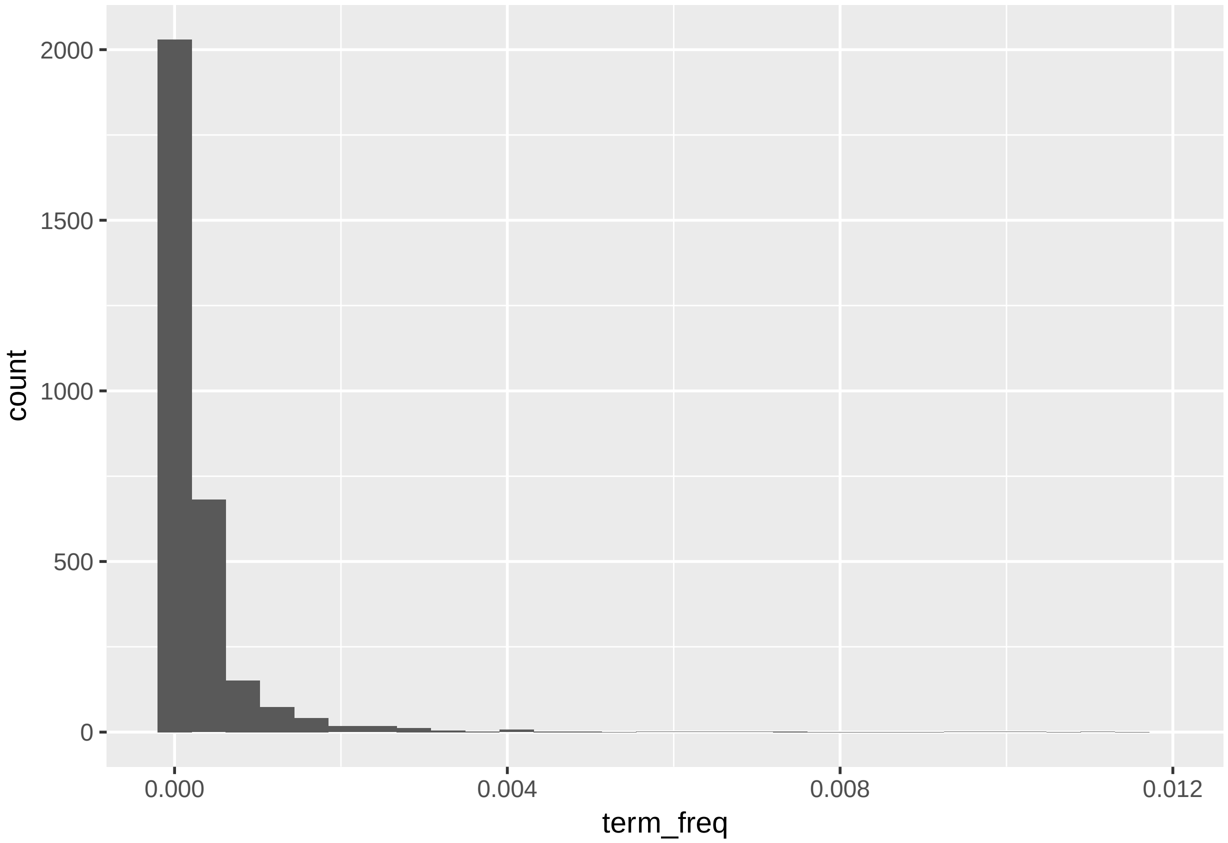
Sometimes, a raw count of a word is less important than understanding how often that word appears in respect to the total number of words in a text. This ratio would be the **term frequency**.

```
doc_words <- doc_words %>%
  mutate(term_freq = n / total)

doc_words
```

Let's graph the term frequency for one of these speeches so we can understand the frequency distribution of words over a text.

```
doc_words %>%
  filter(doc_id == "harry-s-truman-1946.txt") %>%
  ggplot(aes(term_freq)) +
  geom_histogram(show.legend = FALSE) +
  xlim(NA, .012)
```

```
#> `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

#> Warning: Removed 2 rows containing non-finite values (stat_bin).

#> Warning: Removed 1 rows containing missing values (geom_bar).
```



This should make sense. Most words are used relatively rarely in a text. Only a few have a high term frequency.

We could keep filtering this data to see which terms have the high frequency, thus maybe increased significance, for different presidents and different particular speeches. We could also subset based on decade, and get a sense of what was important in each decade. We're going to take a slightly different approach though. We've been looking at term frequency per document. What if we want to know about words that seem more important based on the contents of the entire corpus?

2.3 Tf-idf

For this, we can use term-frequency according to inverse document frequency (tf-idf). Tf-idf meansures how important a word is within a corpus by scaling term frequency per document according to the inverse of the term's document frequency (how many documents within the corpus in which the term appears divided by the number of documents).

We could write our own function for tf-idf, but in this case we'll take advantage of tidytext's implementation.

```
doc_words <- doc_words %>%
  bind_tf_idf(word, doc_id, n)

doc_words
```

The tf-idf value will be:

- lower for words that appear in many documents in the corpus, and lowest when the word occurs in virtually all documents.

- high for words that appear many times in few documents in the corpus, this lending high discrimiatory power to those doucments.

Let's look at some of the words in the corpus that have the highest tf-idf scores, which means words that are particularly distinctive for their documents.

```
doc_words %>%
  select(-total) %>%
  arrange(desc(tf_idf))
```

These results seem appropriate given our history. To understand the occurence of the years we might need to look more closely at the speeches themselves, and determine whether the years are significant or whether they need to be removed from the text. It might be that even if they don't need to be removed from the text overall, they still need to be filtered out within the context of this analysis.

In the same way that we narrowed our analysis to Obama speeches earlier, we could subset the corpus before we calculate the tf-idf score to understand which words are most important for a single president within their sotu speeches. Let's do that for Obama.

```
obama_tf_idf <- tidy_sotu_words %>%
  filter(president == "Barack Obama") %>%
  count(doc_id, word, sort = TRUE) %>%
  bind_tf_idf(word, doc_id, n) %>%
  arrange(desc(tf_idf))

obama_tf_idf
```

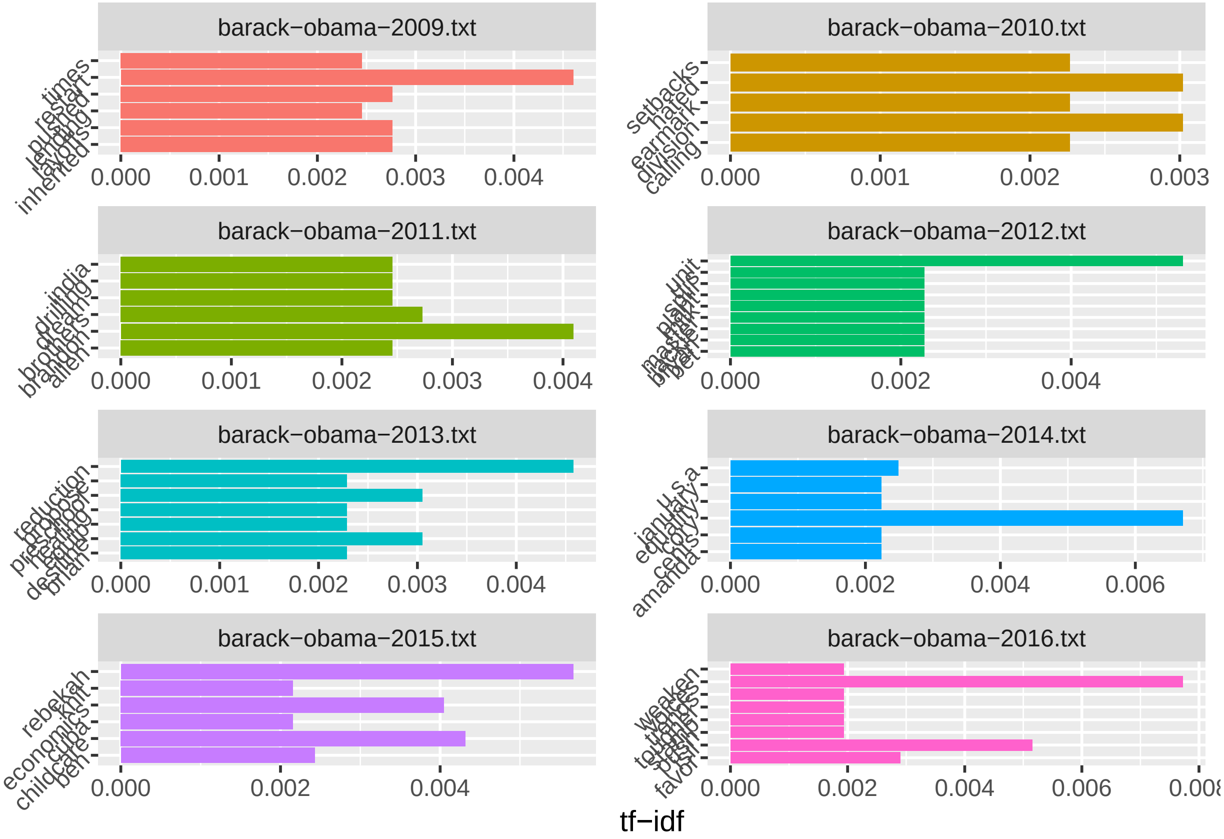
Based on what you know of the Obama years and sotu speeches generally, how would you interpret these results?

Let's try graphing these results, showing the top tf-idf terms per speech for Obama's speeches.

```
obama_tf_idf %>%
  group_by(doc_id) %>%
  mutate(word = factor(word, levels = rev(unique(word)))) %>%
  group_by(doc_id) %>%
  top_n(5) %>%
  ungroup() %>%
  ggplot(aes(word, tf_idf, fill = doc_id)) +
  geom_col(show.legend = FALSE) +
  labs(x = NULL, y = "tf-idf") +
  facet_wrap(~doc_id, ncol = 2, scales = "free") +
  coord_flip() +
  theme(axis.text.y = element_text(angle = 45))
```

```
#> Warning in mutate_impl(.data, dots): Unequal factor levels: coercing to
#> character
```

```
#> Warning in mutate_impl(.data, dots): binding character and factor vector,
#> coercing into character vector

#> Warning in mutate_impl(.data, dots): binding character and factor vector,
#> coercing into character vector

#> Warning in mutate_impl(.data, dots): binding character and factor vector,
#> coercing into character vector

#> Warning in mutate_impl(.data, dots): binding character and factor vector,
#> coercing into character vector

#> Warning in mutate_impl(.data, dots): binding character and factor vector,
#> coercing into character vector

#> Warning in mutate_impl(.data, dots): binding character and factor vector,
#> coercing into character vector

#> Warning in mutate_impl(.data, dots): binding character and factor vector,
#> coercing into character vector

#> Warning in mutate_impl(.data, dots): binding character and factor vector,
#> coercing into character vector

#> Selecting by tf_idf
```



TODO: document-term matrix
TODO: length over time...other similar measures
TODO: variation between the different presidents?
TODO: say something about sentiment analysis and topic modeling