Stat 243: Problem Set 3

Eugene Yedvabny

September 30, 2014

The focus of this assignment was to create a readable and semi-efficient R code for scraping State of the Union speeches and analyzing their properties. I have chosen to rely on the `XML`, `stringR` and `dplyr` packages for the majority of my processing. The code is split up across several functions when the functionality was too much for a one-liner, but a lot of simple counting was just done in-line after the necessary lists were generated.

The initial steps are to download the index and extract all of the links from the HTML table. The list contains 242 entries.

```
# Load in the necessary libraries
library(XML)
library(stringr)
library(lubridate)
library(dplyr,warn.conflicts=FALSE)
library(ggplot2)

# Load in the necessary functions
source("ExtractSpeechContent.R")
source("CleanupSpeech.R")
source("GetSpeechStats.R")

# Access the SOU index page and extract the president, year, and speech URL
sou.url <- "http://www.presidency.ucsb.edu/sou.php"
sou.index <- htmlTreeParse(sou.url, useInternalNodes = TRUE)

# The first entry is the table header, so we can toss it out
speech.links <- xpathSApply(sou.index, "//td[@class='doclist']/a",
                            xmlGetAttr, 'href')[-1]
free(sou.index)
```

The next step is to extract the body and the metadata. That is accomplished using the following function:

```
# Loads in the page from the specified URL
# Extracts the president's name, speech year, and text
# Additionally counts the number of applause and laughter tags
ExtractSpeechContent <- function(speech.url){
  speech.page <- htmlTreeParse(speech.url, useInternalNodes = TRUE)

  # The meta title contains president's name, speech title, and date
  metastring <- speech.page %>% xpathSApply("//meta[@name='title']",
                                            xmlGetAttr, 'content')

  # Using look-ahead and look-behind to extract components
  speech.name <- metastring %>% str_extract(perl("^.*(?=:)"))
  speech.title <- metastring %>% str_extract(perl("(?<=:\\s).*(?=\\s-)"))
  speech.year <- metastring %>% str_extract(perl("(?<=-\\s).*$"))
```

```
  # Get the raw text of the speech
  speech.body <- speech.page %>% xpathSApply("//span[@class='displaytext']",
                                             xmlValue)

  # Return the raw data
  c(speech.name,speech.title,speech.year,speech.body)
}
```

I apply the function onto the list of links to return a list of lists. Since `dplyr` only works on data frames, I transfrom the lists into a data frame. It's a bit messy and `sapply` can return a matrix directly, but turns out that matrix needs to be transposed to come into the right format. I opted for the casting method.

```
# Obtain the speech info from individual HTML pages
speech.data <- lapply(speech.links,ExtractSpeechContent)

# Need to extract columns from the resulting list of lists
speech.data <- matrix(unlist(speech.data), ncol=4, byrow = TRUE)
speech.data <- data.frame(name = speech.data[,1],
                          title = speech.data[,2],
                          date = mdy(speech.data[,3]),
                          speech = speech.data[,4],
                          stringsAsFactors = FALSE)
```

The data frame ends up five columns: president's name, speech title, speech date, and speech body. The `XML` package conveniently removes all the HTML tags. Before the speech is cleaned up, I count the number of Laughter and Applause tags in its body.

```
# Get counts of laughter and applause tags
speech.data <- speech.data %>%
               mutate(
                 num.applause = str_count(speech, perl("(?i)\\[applause\\]")),
                 num.laughter = str_count(speech, perl("(?i)\\[laughter\\]"))
               )
```

The following function will take in the raw speech body and run it through `str_replace_all` to remove dangerous periods and add new-line markers.

```
# Strip out the extraneous content and pretty-print the speech
CleanupSpeech <- function(speech){

  # 1st replace - remove all [] tags
  # 2nd replace - remove all periods after a single letter
  # 3rd replace - remove the periods after Mr. Mrs. or Ms.
  # 4th replace - insert new-line character after every sentence
  speech %>%
  str_replace_all(perl("\\[.*?\\]"),"") %>%
  str_replace_all(perl("(?<=\\s\\w)\\."),"") %>%
  str_replace_all(perl("(Mr|Ms|Mrs)\\."),"\\1") %>%
  str_replace_all(perl("(?<=\\.)\\s"),"\n")
}
```

The function is applied using `mutate` so that the result is appended to the data frame.

```
# Clean-up the speech content
speech.data <- speech.data %>%
               mutate(speech = CleanupSpeech(speech))
```

The splitting into sentences and words doesn't really require its own function since `str_split` is already vectorized. Data frames can't handle lists of lists, so the words and sentences are kept as their own entities. Once they are generated I append the counts of sentences and words to the main data frame.

```
# Split into sentences and words
speech.sentences <- str_split(speech.data$speech,perl("\n"))

speech.words <-  lapply(speech.sentences,
                        function(str_list){
                          unlist(str_split(str_list,perl("(\\s|\n)")))
                        })

#Tack the word counts onto speech data
speech.data <- speech.data %>%
               mutate(
                 num.sentences = sapply(speech.sentences,length),
                 num.words = sapply(speech.words,length)
               )
```

Now comes the fun part - mining the speech content for word frequencies. I wrote a separate function that takes in the speech body and return a list of counts for each of the patterns. When this function is applied on the bodies of all speeches the result is a 242 x 11 matrix (there are 11 patterns).

```
GetSpeechStats <- function(speech){

  # RegEx patterns to look for
  patterns <- c("\\s(I)\\s",
                "(?i)\\s(we)\\s",
                "America(n)*",
                "(?i)Democra(t|cy|tic)+",
                "(?i)Republic(an)*",
                "(?i)Free(dom)*",
                "\\s(war)",
                "(?i)\\sGod(?!\\sbless)",
                "(?i)\\sGod\\sBless",
                "(?i)(Jesus|Christ)",
                "(?i)econom"
              )

  # Sub-function for calling within sapply
  ProcSpeech <- function(pattern,speech){
    str_count(speech,perl(pattern))
  }

  # Call str_count for all patterns and on all speeches
  sapply(patterns,ProcSpeech,speech)
}
```

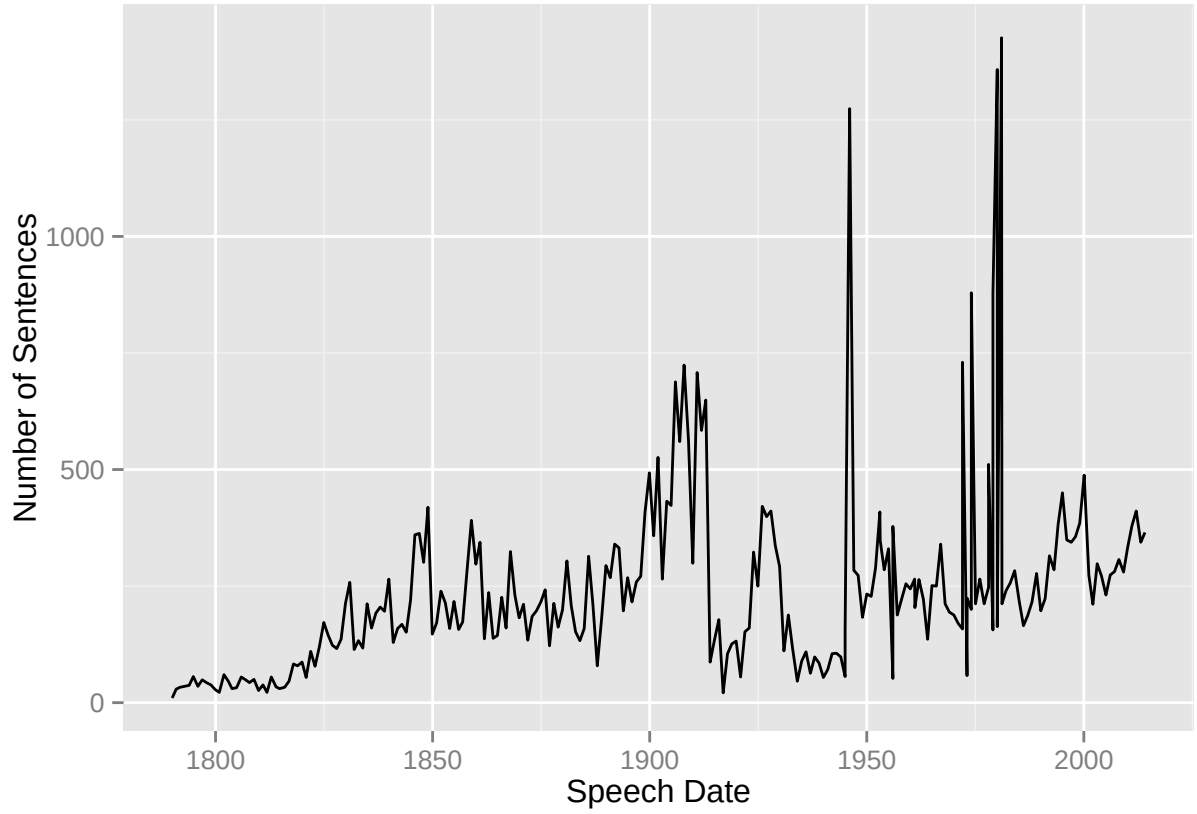
```
speech.freq.counts <- data.frame(GetSpeechStats(speech.data$speech))
speech.data <- cbind(speech.data,speech.freq.counts)
```

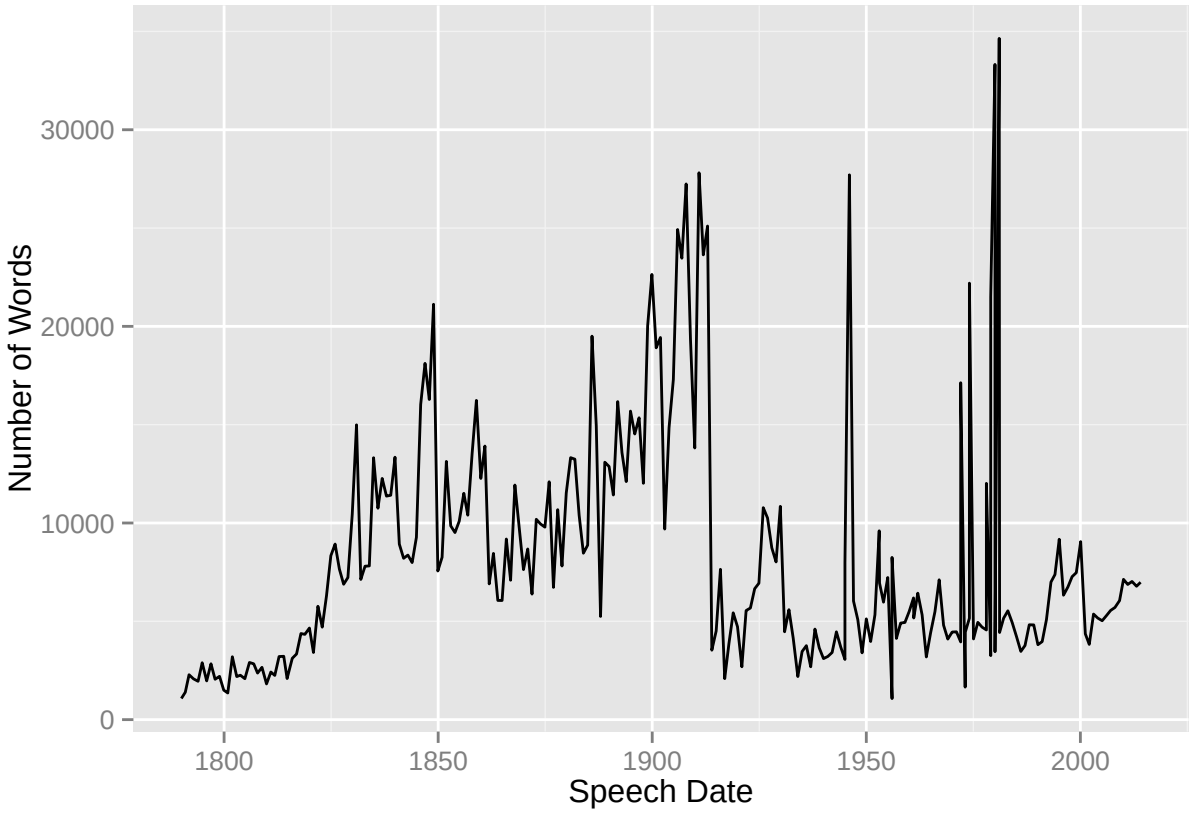
With all the frequencies compiled, time to plot some interesting trends.

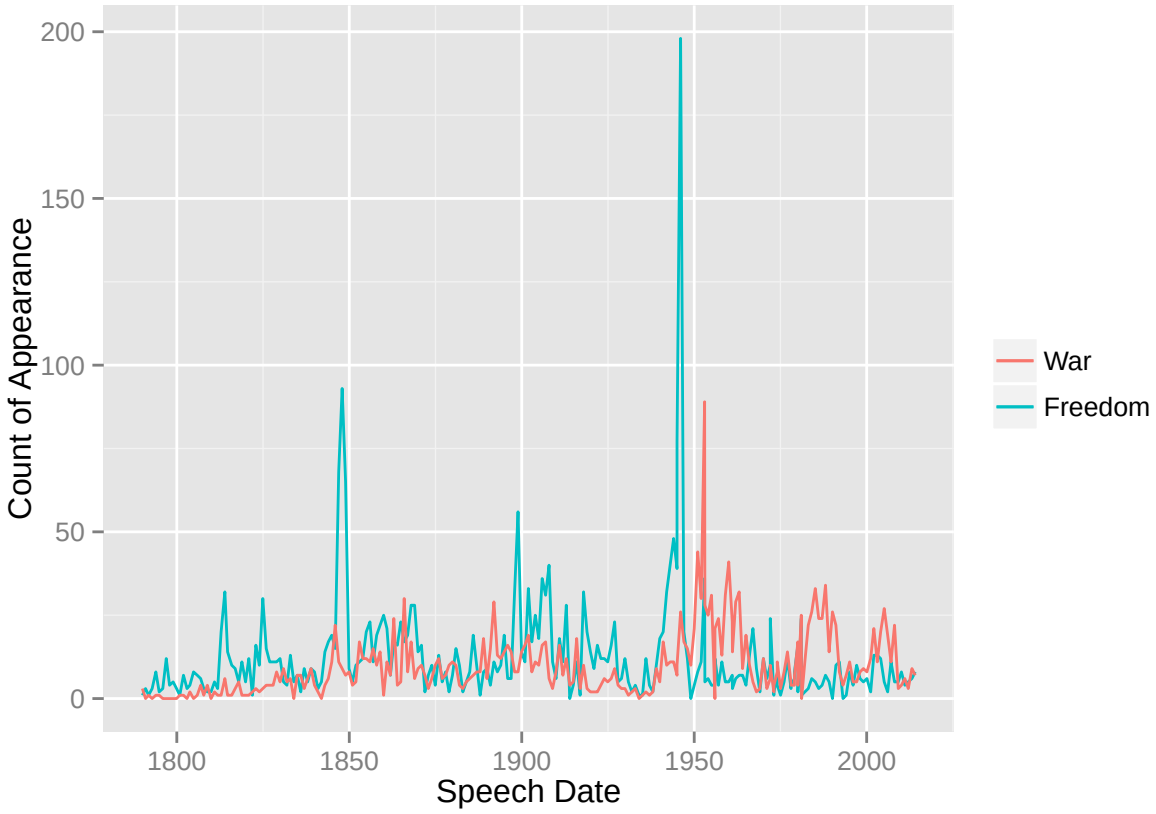
```
ggplot(speech.data,aes(x=date,y=num.sentences)) +
  geom_line() +
  xlab("Speech Date") +
  ylab("Number of Sentences")

ggplot(speech.data,aes(x=date,y=num.words)) +
  geom_line() +
  xlab("Speech Date") +
  ylab("Number of Words")

ggplot(speech.data,aes(x=date)) +
  geom_line(aes(y=X.s.war.,color='red')) +
  geom_line(aes(y=X..i.Free.dom..,color='blue')) +
  scale_colour_discrete(name="",labels=c("War","Freedom")) +
  xlab("Speech Date") +
  ylab("Count of Appearance")
```




30000
20000
10000
0
Number of Words
1800
1850
1900
1950
2000
Speech Date



Unfortunately at this stage I ran out of time and could not complete the Democrat vs Republican comparison. But it should be fairly trivial by assigning the presidents to political parties and then using `group_by` to look at the trends.

This concludes my foray into analyzing State of the Union speeches.