Homework #6 — First Presidential Debate 2020

Stan Doan

10/4/2020

Problem 1

See `data_munge.R` below.

Problem 2

(a) `get_word_counts(d, speaker)`

I wrote two other functions, one to split statements into words, `get_split_sen(d, speaker)`, and one to "clean" those words, `clean_words(words)`. The reason for the latter is to count different forms and tenses of one word as one word. For example, *prosecute*, *prosecuted*, and *prosecuting* will be uniformly counted as *prosecute*. This process also helps reduce unnecessary counting, especially for words like *doesn't*, *I'll*, *they're*, which are in fact *do*, *I*, and *they*, respectively. The variants of *be*, like *are*, *wasn't*, *being*, and *been*, also make a good example as to how cleaning improves my answer to Problem 3.

```
debate <- get_debate()
head(get_word_counts(debate, "biden"))
```

```
##         word count
## 3          A   127
## 4    ABILITY     2
## 5       ABLE    17
## 6      ABOUT    41
## 7 ABSOLUTELY     3
## 8     ABSORB     1
```

(b) `total_word_counts(d, speakers)`

```
bidenWallace <- total_word_counts(debate, c("Wallace", "Biden")) %>%
  arrange(desc(count))
head(bidenWallace)
```

```
##   word count
## 1  THE   557
## 2   BE   437
## 3   TO   394
## 4  YOU   316
## 5  AND   259
## 6   OF   234
```

Problem 3

Cleaning Data (`clean_words(words)`)

I use a dataset of English verb forms (`raw_data/english_verbs.csv`) to identify verbs in the transcript and convert their conjugations into original form. A more advanced approach includes different forms of nouns and adjectives, but I haven't got there yet.

```
words <- c("Apple", "faked", "been", "went", "they're")
clean_words(words)
```

```
## [1] "APPLE" "BE"    "FAKE"  "THEY"  "GO"
```

Ranking Words

There are four points of reference in my word-ranking algorithm: Trump, Biden, Wallace, and the English speakers. Words the first three said are recorded in the transcript, and the most common English words are given in `word_frequency.csv`. I extract these data and store them in a frame with three columns: `speaker`, `word`, and `count`. I then employ the *term frequency — inverse document frequency* method in package `tidytext` to rank words based on (1) how often they are spoken (higher is better), and (2) how many "people" (out of four) often speak them (lower is better). In other words, words are deemed more characteristic of a person if he/she uses them more frequently, and less so if they are also used by other people.

After refining word data and contrasting speakers' lexicons with each other, here is the results.

```
whatBidenSaid <- prepare_word_cloud(debate, "biden")
head(whatBidenSaid)
```

```
##               word       weight
## 22610    DISCREDIT 0.0014913263
## 24100   AFFORDABLE 0.0010652331
## 38100      AMERICA 0.0008521865
## 9310  BILLIONAIRES 0.0006391399
## 26010         EMIT 0.0006391399
## 30210          FBI 0.0006391399
```

```
whatTrumpSaid <- prepare_word_cloud(debate, "trump")
head(whatTrumpSaid)
```

```
##         word       weight
## 539 NOVEMBER 0.0013747075
## 348   HAPPEN 0.0011003564
## 128    CHINA 0.0009819340
## 515   MOSCOW 0.0009819340
## 419  JOHNSON 0.0007855472
## 466   LOWEST 0.0007855472
```

The wordclouds.

```
bidenWC <- wordcloud2(whatBidenSaid, size = 1.5, color = "random-light") %>%
  saveWidget("../images/biden.html", selfcontained = F)
webshot("../images/biden.html", "../images/biden.png", delay = 5,
        vwidth = 2000, vheight = 1000)
```


```
trumpWC <- wordcloud2(whatTrumpSaid, size = 1.5, color = "random-dark") %>%
  saveWidget("../images/trump.html", selfcontained = F)
webshot("../images/trump.html", "../images/trump.png", delay = 5,
        vwidth = 2000, vheight = 1000)
```

Weighting

Without Wallace

As a moderator, Wallace is probably the least relevant among the four points of reference. Here's the results when I filter out Wallace.

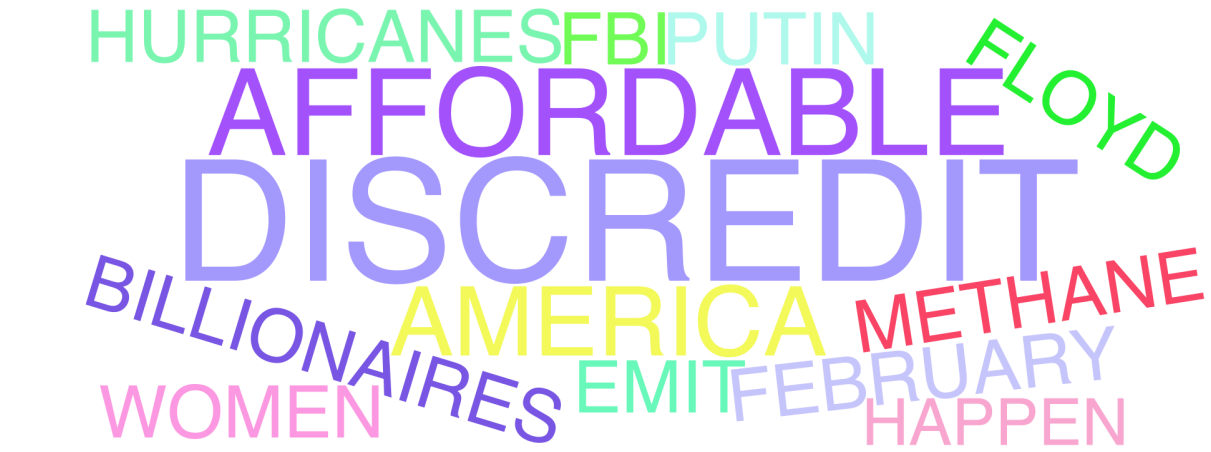
```
whatBidenSaid <- prepare_word_cloud(debate, "biden", wallace = F)
bidenWC <- wordcloud2(whatBidenSaid, size = 1.5, color = "random-light") %>%
  saveWidget("../images/biden_without_wallace.html", selfcontained = F)
webshot("../images/biden_without_wallace.html", "../images/biden_without_wallace.png",
        delay = 5, vwidth = 2000, vheight = 750)
```


```
whatTrumpSaid <- prepare_word_cloud(debate, "trump", wallace = F)
trumpWC <- wordcloud2(whatTrumpSaid, size = 1.5, color = "random-dark") %>%
  saveWidget("../images/trump_without_wallace.html", selfcontained = F)
webshot("../images/trump_without_wallace.html", "../images/trump_without_wallace.png",
        delay = 5, vwidth = 2000, vheight = 750)
```

Change Weight of the English speakers

Set weight of the most common words `weightCommon` to 0.00001. I think this method yields the best results, as most trivial words, like *sure* and *man*, are removed, but important common words like *destroy* are kept.

```
whatBidenSaid <- prepare_word_cloud(debate, "biden", weightCommon = 0.00001)
bidenWC <- wordcloud2(whatBidenSaid, size = 1.5, color = "random-light") %>%
  saveWidget("../images/biden_deweight_common.html", selfcontained = F)
webshot("../images/biden_deweight_common.html", "../images/biden_deweight_common.png",
        delay = 5, vwidth = 2000, vheight = 750)
```




```
whatTrumpSaid <- prepare_word_cloud(debate, "trump", weightCommon = 0.00001)
trumpWC <- wordcloud2(whatTrumpSaid, size = 1.5, color = "random-dark") %>%
  saveWidget("../images/trump_deweight_common.html", selfcontained = F)
webshot("../images/trump_deweight_common.html", "../images/trump_deweight_common.png",
        delay = 5, vwidth = 2000, vheight = 750)
```

Code

data_munge.R

```
#Transcipt to Dataset
transcript <- file("../processed_data/Presidential_Debate_Transcript_processed.txt", "r")
lines <- readLines(transcript)

d <- data.frame(linenumber = "2", speaker = "WALLACE", statement = lines[2],
                time = "1min20sec", stringsAsFactors = F)
spkrNames <- c("WALLACE", "TRUMP", "BIDEN")

for (i in seq(5, length(lines), 3)) {
  spkrDetect <- str_detect(lines[i-1], c("Wallace: ", "Trump: ", "Biden: "))
  thisSpeaker <- spkrNames[ which(spkrDetect %in% TRUE) ]

  timestamp <- str_sub(lines[i-1], str_locate(c(lines[i-1]), "\\(")[1]+1,
                       str_locate(lines[i-1], "\\)")[1])
  str_replace(timestamp, ":", "XX") %>% str_replace("\\)","sec") -> timestamp
  if (all(str_detect(timestamp, c("XX",":")))) {
    str_replace(timestamp, "XX", "hour") %>% str_replace(":", "min") -> timestamp
  } else str_replace(timestamp, "XX", "min") -> timestamp

  d[nrow(d) + 1,] <- c(paste(i), thisSpeaker, lines[i], timestamp)
}

write.csv(d, "../processed_data/pres_debate.csv", row.names = F)

#Word Frequency
word_freq <- read.csv("../raw_data/word_frequency.csv", header = T, stringsAsFactors = F)

select(word_freq, -Rank, -Part.of.speech, -Dispersion) -> word_freq
names(word_freq) <- c("word", "count")
dplyr::mutate_all(word_freq, funs(toupper)) -> word_freq
word_freq$count <- as.integer(word_freq$count)
str_remove_all(word_freq$word, "[[:space:]]") -> word_freq$word

write.csv(word_freq, "../processed_data/word_frequency_processed.csv", row.names = F)

#English Verbs
verbs <- read.csv("../raw_data/english_verbs.csv", header = F,
                  stringsAsFactors = F, fileEncoding="latin1")

c("v1", "v2", "v3", "v4", "v5") -> names(verbs)
dplyr::mutate_all(verbs, funs(toupper)) -> verbs

write.csv(verbs, "../processed_data/english_verbs_processed.csv", row.names = F)
```

configuration.R

```
library(wordcloud2)
library(webshot)
library(htmlwidgets)
library(tidyverse)
library(magrittr)
```

```
library(tidytext)

source("data.R")
source("analysis.R")
```

data.R

```
get_debate <- function() {
  d <- read.csv("../processed_data/pres_debate.csv", header = T, stringsAsFactors = F)
  return(d)
}

get_word_freq <- function() {
  d <- read.csv("../processed_data/word_frequency_processed.csv", header = T,
                stringsAsFactors = F)
  return(d)
}

get_verbs <- function() {
  d <- read.csv("../processed_data/english_verbs_processed.csv", header = T,
                stringsAsFactors = F)
  return(d)
}
```

analysis.R

```
get_word_counts <- function(d, speaker) {
  splitSen <- get_split_sen(d, speaker)

  wordCounts <- as.data.frame(table(splitSen), stringsAsFactors = F)
  c("word", "count") -> names(wordCounts)

  return(wordCounts[3:nrow(wordCounts),])
}

total_word_counts <- function(d, speakers) {
  splitSen <- get_split_sen(d, speakers[1])
  for (i in 2:length(speakers))
    splitSen <- c(splitSen, get_split_sen(d, speakers[i]))

  wordCounts <- as.data.frame(table(splitSen))
  c("word", "count") -> names(wordCounts)
  return(wordCounts[3:nrow(wordCounts),])
}

prepare_word_cloud <- function(d, speaker, trump = T, biden = T,
                               wallace = T, common = T, weightCommon = 1) {
  speakr <- speaker
  dplyr::filter(word_freq_data(d, trump, biden, wallace, common, weightCommon),
                speaker == toupper(speakr)) %>%
    dplyr::select(-speaker, -tf, -idf, -count) -> out
  c("word", "weight") -> names(out)
  return(out)
}
```

```
get_split_sen <- function(d, speaker) {
  speakr <- speaker
  speakerSen <- dplyr::filter(d, speaker == toupper(speakr))$statement
  splitSen <- unlist(strsplit(speakerSen, split=" "))
  return(clean_words(splitSen))
}

clean_words <- function(words) {
  get_verbs() -> verbs
  toupper(words) %>%
    str_remove_all("[[:punct:]&&[^']]|\\"|\\"|\\...|CROSSTALK|[[:digit:]]|[[:space:]]
                   |N'T|'S|'M|'RE|'LL|'VE|'D|BIDEN|CHRIS|JOE") %>%
      sort() -> words

  for(i in 2:5) {
    which(verbs[[i]] %in% words) -> thisTense
    for(j in 1:length(thisTense)) {
      which(words %in% verbs[[i]][thisTense[j]]) -> thesePositions
      words[thesePositions] <- verbs[[1]][thisTense[j]]
    }
  }
  return(words)
}

word_freq_data <- function(d, trump = T, biden = T, wallace = T, common = T, weightCommon = 1) {
  trumpWords <- ifelse(trump, get_words(d, "trump"))
  bidenWords <- ifelse(biden, get_words(d, "biden"))
  wallaceWords <- ifelse(wallace, get_words(d, "wallace"))
  commonWords <- ifelse(common, data.frame(speaker = "COMMON",
                                           change_weight(get_word_freq(), weightCommon),
                                           stringsAsFactors = F))

  allWords <- rbind(ifelse(trump, trumpWords), ifelse(biden, bidenWords),
                    ifelse(wallace, wallaceWords), ifelse(common, commonWords)) %>%
    bind_tf_idf(word, speaker, count) %>%
      arrange(desc(tf_idf))
  return(allWords)
}

get_words <- function(d, spkr) {
  return(data.frame(speaker = toupper(spkr), get_word_counts(d, spkr), stringsAsFactors = F))
}

change_weight <- function(d, newWeight = 1) {
  d$count <- d$count * newWeight
  return(d)
}

ifelse <- function(yesno, yay) {
  if(yesno) return(yay)
  else return(NA)
}
```