Keyword analysis of State of the Union Speeches

Jerid Francom

2015-04-04

Not long ago I gave an introduction to text analysis and data mining for the Wake Forest University community. In that tutorial I demonstrated how to do a basic keyword analysis on "State of the Union" speeches from 1947 to present using the freeware AntConc. Although AntConc is a fantastic resource, it's not my go-to approach for doing text analysis. In this post I will re-create the analysis using R: reading, exploring, and cleaning data as well as the main keyword analysis and corresponding visualizations.

The first step is to read in the `sotu` text files we scraped from the web.

```
library(tm)
library(dplyr)

# `read.corpus` function
read.corpus <- function(directory, pattern = "", to.lower = TRUE) {
  corpus <- DirSource(directory = directory, pattern = pattern) %>%
    VCorpus # Read files and create `VCorpus` object
  if(to.lower == TRUE) corpus <- # Lowercase text
    tm_map(corpus,
           content_transformer(tolower))
  return(corpus)
}
```

In this code chunk we use the `tm` package to read in files in a specific directory whose file names match a certain pattern, then lowercase the text. Since we want to create corpora from file names matching `republican` on the one hand and `democrat` on the other, I chose to create a function as not to be too redundant with the code. Inside the function `read.corpus()`, we use the `tm::DirSource()` function to point R to the directory where the `sotu` texts are listed and then takes a `pattern` to match file names. This result is handed off to `VCorpus()` which creates a `tm` package `VCorpus` object.[1] I then include an option to lowercase the text. In this case we will want our text lowercased, so I've set it to the default in the function (`to.lower = TRUE`), but that's not always the case.

Now apply the function to get the Republican and Democrat texts and confirm that we've got `VCorpus` objects.

```
rep.corpus.raw <- read.corpus(directory = "texts/", pattern = "republican")
  class(rep.corpus.raw)
```

```
## [1] "VCorpus" "Corpus"
```

```
dem.corpus.raw <- read.corpus(directory = "texts/", pattern = "democrat")
  class(rep.corpus.raw)
```

```
## [1] "VCorpus" "Corpus"
```

Let's do a quick inspection of the results. It's good to know a few functions for exploring `VCorpus` objects.

[1] The handing off portion uses the `%>%` syntax from the `magrittr` package (loaded through the `dplyr` package here).

```
rep.corpus.raw %>%
  print() # an overview of the object
```

```
## <<VCorpus (documents: 34, metadata (corpus/indexed): 0/0)>>
```

```
rep.corpus.raw %>%
  meta(tag = "id") %>% # a list of the file names in the corpus
  unlist()
```

```
##  [1] "republican-1954.txt" "republican-1955.txt" "republican-1956.txt"
##  [4] "republican-1957.txt" "republican-1958.txt" "republican-1959.txt"
##  [7] "republican-1960.txt" "republican-1970.txt" "republican-1971.txt"
## [10] "republican-1972.txt" "republican-1974.txt" "republican-1975.txt"
## [13] "republican-1976.txt" "republican-1977.txt" "republican-1981.txt"
## [16] "republican-1982.txt" "republican-1983.txt" "republican-1984.txt"
## [19] "republican-1985.txt" "republican-1986.txt" "republican-1987.txt"
## [22] "republican-1988.txt" "republican-1989.txt" "republican-1990.txt"
## [25] "republican-1991.txt" "republican-1992.txt" "republican-2001.txt"
## [28] "republican-2002.txt" "republican-2003.txt" "republican-2004.txt"
## [31] "republican-2005.txt" "republican-2006.txt" "republican-2007.txt"
## [34] "republican-2008.txt"
```

So `print()` provides us a basic overview of the number of documents in the corpus. `meta()` with the `tag` set to `"id"` returns the file names themselves. This is good here to verify that we've in fact captured the appropriate texts in each corpus. `content(rep.corpus.raw)` will return all of the text in the corpus. Go ahead and try it at home –I'll save some screen real estate and skip this command.

In a keyword analysis we will be working with, well, words, and their frequencies so let's make a first pass at creating a word frequency list.

```
create.wordlist <- function(corpus, create_df = FALSE) {
  wordlist <- corpus %>%
    DocumentTermMatrix() %>%
    as.matrix() %>%
    colSums()
  if(create_df == TRUE) wordlist <- wordlist %>%
    data.frame(words = names(.), freq = ., row.names = NULL)
  return(wordlist)
}
```

Again, I turn to creating a function because we want to reduce redundancy. This function, `create.wordlist()`, takes a `VCorpus` object (corpus) as an argument and provides an option to output the wordlist as a `data.frame` (create_df). The `corpus` is passed to `tm::DocumentTermMatrix()` which is another `tm` object, but one that holds the corpora in matrix form. It's worth taking a look at a simplified example to give you an idea what we are dealing with.

```
documents <- c("this is a sentence", "here is another one", "now another")
dtm <- VectorSource(x = documents) %>%
  VCorpus %>%
  DocumentTermMatrix
```

So the three character vectors in `documents` is read into `VCorpus` format and then sent to `DocumentTermMatrix`. The result in `dtm` can be `inspect`ed:

```
dtm %>%
  inspect
```

```
## <<DocumentTermMatrix (documents: 3, terms: 6)>>
## Non-/sparse entries: 7/11
## Sparsity           : 61%
## Maximal term length: 8
## Weighting          : term frequency (tf)
##
##     Terms
## Docs another here now one sentence this
##    1       0    0   0   0        1    1
##    2       1    1   0   1        0    0
##    3       1    0   1   0        0    0
```

So looking at the bottom portion, the `DocumentTermMatrix` contains a matrix where rows correspond to each of the character vectors in `documents` and the columns each of the unique words. This matrix summarizes the word frequency and dispersion contained in our text documents.

Returning to the `creat.wordlist()` function from above, we then remove some of the extra bells and whistles in the `DocumentTermMatrix` with `as.matrix()` –giving us just the matrix that we are interested in. Summing the columns for each word (`colSums()`) gives us the word frequencies. Again I include an option to change the output to another data type, in this case a tabular, excel-like format – a.k.a. the `data.frame`.[2]

Run the `create.wordlist()` function, check the class, and find out how many (unique) words are in each of our corpora.

```
rep.wordlist <- create.wordlist(corpus = rep.corpus.raw, create_df = TRUE)
  class(rep.wordlist)
```

```
## [1] "data.frame"
```

```
  nrow(rep.wordlist)
```

```
## [1] 18753
```

```
dem.wordlist <- create.wordlist(corpus = dem.corpus.raw, create_df = TRUE)
  class(dem.wordlist)
```

```
## [1] "data.frame"
```

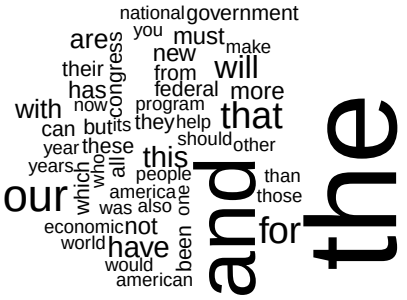
```
  nrow(dem.wordlist)
```

```
## [1] 19731
```

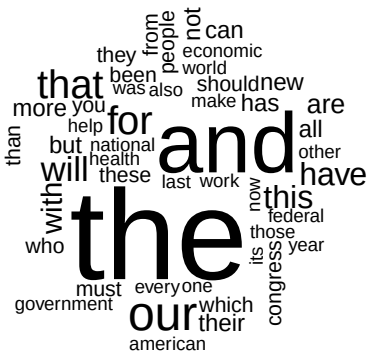
I fun way to visualize a word frequency list is via word clouds. Conveniently, there's an R package for that, appropriately named `wordcloud`.

[2]Passing, or 'piping' output using the `%>%` syntax assumes that the first argument of the next function will take the output of the previous. If that is not the case, then the `.` operator can be used to specify where insert the input.

```
library(wordcloud)
wordcloud(words = rep.wordlist$words, freq = rep.wordlist$freq, max.words = 50)
```




```
wordcloud(words = dem.wordlist$words, freq = dem.wordlist$freq, max.words = 50)
```



`wordcloud::wordcloud()` takes the words and their frequencies, minimally. But you can also specify a number of other parameters to modify the content and format of the word cloud. Here I've limited the output to the top 50 words.

Not what you expected? It's real easy to overlook the fact that the most frequent words in a given language are often the least interesting, from a content standpoint. These words are often closed-class words (determiners, prepositions, conjunctions, etc.) and contrast to open-class words (nouns, verbs, adjectives) that are usually more interesting –especially for us at the moment. So we are going to want to filter them. But how do you know what words to filter? Enter "stopword lists": pre-compiled lists of the kinds of words we are looking to eliminate.

That's area we need to address. Other standard filters include removing punctuation, numerals, and extra whitespace. In addition, on inspecting the word frequency lists it appears that there are certain words enclosed inside brackets `[]` and parenthesis `()` that should be deleted as well given they correspond, by and large, to meta discourse not the content of the speeches. See below[3]:

```
rep.wordlist$words %>%
  grep(pattern = "[\\(\\[\\]+", x = ., value = TRUE)
```

```
##  [1] "(\"cds\")"          "($14.7"             "(1)"
##  [4] "(1980)"             "(2)"                "(3)"
##  [7] "(4)"                "(5)"                "(6.9%"
## [10] "(asean)"            "(at"                "(cdbg)"
## [13] "(ceta)."            "(chap)"             "(eeoc)"
## [16] "(fao),"             "(infce),"           "(nhsc)"
```

[3] Regular expressions are used for pattern matching

```
## [19] "(ofccp)"            "(the"               "(tip)"
## [22] "(udag)"             "(wic)"              "[$.5"
## [25] "[$]1.1"             "[50,000]"           "[act]"
## [28] "[africa],"          "[applause]"         "[applause]third,"
## [31] "[arkansas],"        "[as]"               "[at"
## [34] "[delivered"         "[former"            "[full]"
## [37] "[in"                "[iowa],"            "[it]"
## [40] "[last]"             "[laughter]"         "[laughter]-and"
## [43] "[laughter]-but"     "[laughter]."        "[licensing]"
## [46] "[moment"            "[national"          "[of]"
## [49] "[oil]"              "[read"              "[south"
## [52] "[the]"              "[un]"               "[washington]"
## [55] "401(k)"             "8(a)"               "crisis-[laughter]-"
## [58] "last--maybe--["
```

Let's clean up the corpus. Removing bracket/ parenthetical words, stopwords, punctuation, numerals, and whitespace. The `tm` package includes the `tm_map()` function that provides many easy-to-implement ways to transform the corpus content.[4]

```
clean.corpus <- function(corpus) {
  # Clean a `tm` corpus #
  # Substitution function for to simplify `tm_map`
  my.sub <- content_transformer(
    function(x, pattern, replacement)
      gsub(pattern, replacement, x))

  corpus %>%
    tm_map(my.sub, "\\[\\w+\\]", "") %>% # ex.`[applause]`
    tm_map(my.sub, "\\(\\w+\\)", "") %>% # ex. `(tip)`
    tm_map(removeWords, stopwords("english")) %>% # english stopwords
    tm_map(removePunctuation, preserve_intra_word_dashes = TRUE) %>% # keep `low-income`
    tm_map(removeNumbers) %>% # numerals
    tm_map(stripWhitespace) # clean up any trailing whitespace
}
```

Now apply the filters and clean this corpus up!

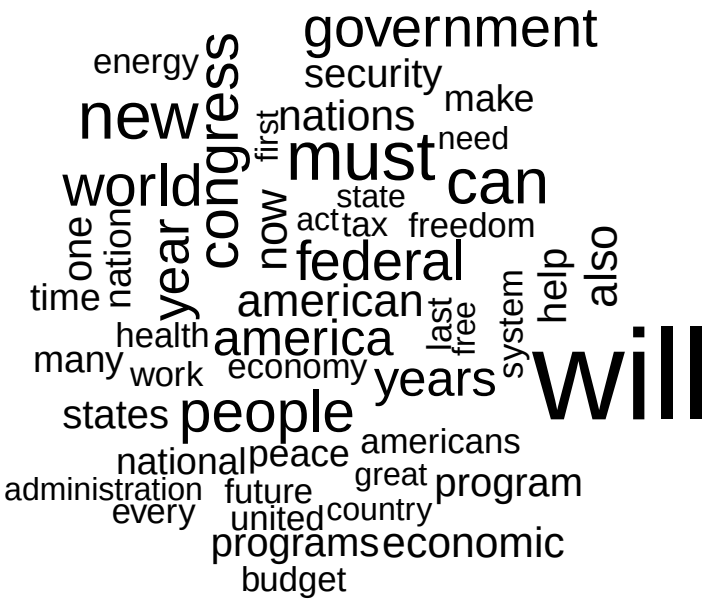
```
rep.corpus <- clean.corpus(corpus = rep.corpus.raw)
dem.corpus <- clean.corpus(corpus = dem.corpus.raw)
```

Returning to our word clouds. We need to create new wordlists from the cleaned data and then visualize the results.

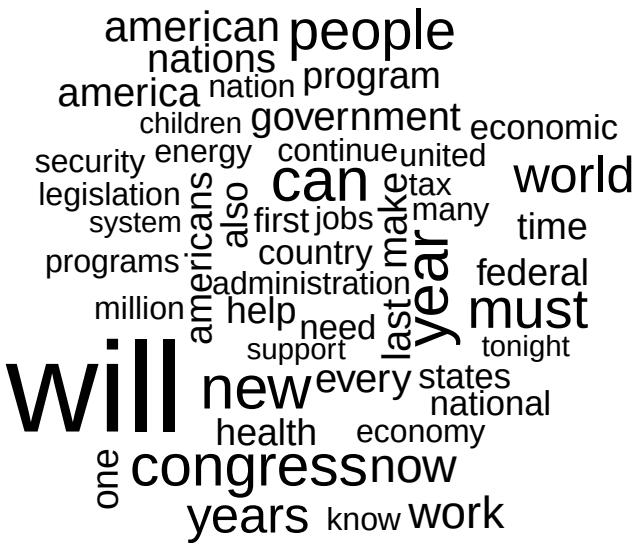
```
rep.wordlist <- create.wordlist(corpus = rep.corpus, create_df = TRUE)
dem.wordlist <- create.wordlist(corpus = dem.corpus, create_df = TRUE)

wordcloud(words = rep.wordlist$words, freq = rep.wordlist$freq, max.words = 50)
```

[4] One twist is the `my.sub()` function I wrote to simplify adding custom transformations to `tm_map()`.




```
wordcloud(words = dem.wordlist$words, freq = dem.wordlist$freq, max.words = 50)
```



Alright, now we are in a position to do some comparing. Looking at the word clouds we can see obvious similarities and differences. The size of the text, however, can be misleading here: text between each word cloud is not on the same scale. That is, it's hard to know how these frequencies/text sizes really match up between the corpora. To identify those words that are most/ least indicative of each corpus (when compared to the other corpus) we want to compare the relative frequency of words in the `rep.corpus` with the relative frequencies in the `dem.corpus`. Using the ratio of `rep.corpus`/ `dem.corpus` we will find that words with highest values tend to be more indicative of the `rep.corpus` and lowest values more indicative of the `dem.corpus`. This calculation is known as a Relative Frequency Ratio (Damerau, 1993).

```
rfr.calc <- function(wordlist.a, wordlist.b, create_df = FALSE) {
  # Calculate Relative Frequency Ratio #
  # Words in wordlist.b, not in wordlist.a
  wordlist.a[setdiff(names(wordlist.b), names(wordlist.a))] <- 0 # Fill missing words
  # Words in wordlist.a, not in wordlist.b
```

```
  wordlist.b[setdiff(names(wordlist.a), names(wordlist.b))] <- 0 # Fill missing words

  vocabulary <- wordlist.a %>% names # get complete vocabulary listing

  a.ratios <- wordlist.a/sum(wordlist.a) # ratios for wordlist a
  b.ratios <- wordlist.b/sum(wordlist.b) # ratios for wordlist b

  # wordlist.a ratio/ wordlist.b ratio (with +1 smoothing)
  rfr <- log((a.ratios[vocabulary] + 1) / (b.ratios[vocabulary] + 1))

  # Coerce to a `data.frame`
  if(create_df == TRUE) rfr <- rfr %>%
    data.frame(Words = names(.),
               Scores = .,
               row.names = NULL,
               stringsAsFactors = FALSE) %>%
    arrange(Scores)
  return(rfr)
}
```

The function `rfr()` implements the Relative Frequency Ratio calculation with a couple practical additions. First, each wordlist will contain words that do not occur in the other. Since this will lead to undefined ratios, a single value is added to each ratio calculation (+1 smoothing). Furthermore, I've taken the `log()` of this ratio to provide a more appeasing visual distribution (see below). The results are optionally returned as a `data.frame`.

First, we need to create wordlists from our cleaned-up corpora. Then run the `rfr()` function.

```
rep.wordlist <- create.wordlist(corpus = rep.corpus)
dem.wordlist <- create.wordlist(corpus = dem.corpus)

rfr_df <- rfr.calc(wordlist.a = rep.wordlist,
                   wordlist.b = dem.wordlist,
                   create_df = TRUE)
```

Let's look at the first lines of `rfr_df`.

```
rfr_df %>%
  head
```

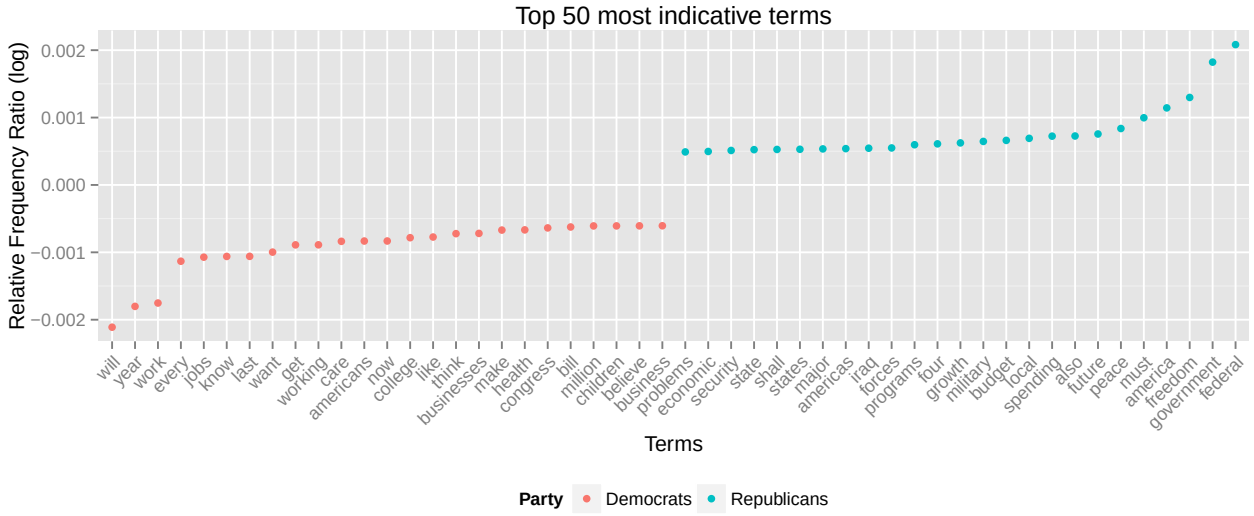
```
##   Words       Scores
## 1  will -0.002111112
## 2  year -0.001802240
## 3  work -0.001752393
## 4 every -0.001132055
## 5  jobs -0.001071891
## 6  know -0.001060919
```

Looks good. Let's take the top 25 and the bottom 25 and add Party labels to get ready to visually summarize these top results.

```
# Get highest and lowest scores (i.e. Republican/ Democrat)
rep.dem_df <- rbind(head(rfr_df, 25), tail(rfr_df, 25))
rep.dem_df$Party <- ifelse(test = (rep.dem_df$Scores > 0),
                           yes = "Republicans",
                           no = "Democrats")
```

`rbind()` takes the `head()`, or top 25 and the `tail()` or bottom 25 and binds them by row. Since we know that the point at which the ratio is equal, that is, Republican words and Democrat words have the same relative frequency is 1 and `log(1)` is 0, then values 0 and above are republican-indicative otherwise democrat-indicative. The `ifelse()` statement tests this and applies the Party labels accordingly.

```
library(ggplot2)
ggplot(rep.dem_df, aes(x = reorder(Words, Scores), y = Scores, color = Party)) +
  geom_point() +
  theme_gray() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  xlab("Terms") + ylab("Relative Frequency Ratio (log)") + ggtitle("Top 50 most indicative terms") +
  theme(legend.position = "bottom")
```



Very cool. One more step that would really complement this analysis would be to do a concordance on terms of interest to see particular words in context. Below is a more complex function that is a first stab at doing just that.

```
concordance <- function(texts, pattern, margin = 30, n = 0) {
  # A script to create a concordance listing for character vectors #
  pattern <- tolower(pattern) # lowercase
  concordance <- lapply(tolower(texts), function(i) { # loop over lowercased texts
    str_locate_all(string = i, # find all indices for pattern matches
                   pattern = pattern) %>%
      sapply(function(j) { # loop over index pairs (start/ end)
        str_sub(string = i,
                start = as.integer(j[, 'start'])-margin, # extract the pattern
                end = as.integer(j[, 'end'])+margin) %>% # with margins
          str_replace_all(pattern = pattern, # replace pattern with padded pattern
                          replacement = str_pad(string = pattern,
                                                width = nchar(pattern)+4,
```

```
                                                  side = "both",
                                                  pad = " "))
      }) %>%
      unlist # clean up output
  }) %>%
    unlist %>% # clean up output
    as.vector  # clean up output
  if(n > 0) concordance <- concordance[1:n]
  return(concordance)
}
```

So to explore the word "federal" in the `rep.corpus` we run:

```
concordance(texts = content(rep.corpus.raw), pattern = "federal", margin = 35, n = 5)
```

```
## [1] "tion in the armed forces and other   federal   activities is on the way out. we h"
## [2] "oyees have been separated from the   federal   government. our national security "
## [3] "pecial employment standards of the   federal   government i turn now to a matter "
## [4] "e government. it recognizes that a   federal   budget should be a stabilizing fac"
## [5] "ons for some crops and apply rigid   federal   controls over the use of the diver"
```

And there you have it a re-creation of the analysis using AntConc, this time using R.

```
sessionInfo()
```

```
## R version 3.1.3 (2015-03-09)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: OS X 10.10.2 (Yosemite)
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] graphics  grDevices utils     datasets  stats     methods   base
##
## other attached packages:
## [1] wordcloud_2.5      RColorBrewer_1.1-2 stringr_0.6.2
## [4] dplyr_0.4.1        tm_0.6             NLP_0.1-6
## [7] Rdym_0.2.0         ggplot2_1.0.1
##
## loaded via a namespace (and not attached):
##  [1] assertthat_0.1   colorspace_1.2-5 DBI_0.3.1        digest_0.6.8
##  [5] evaluate_0.5.5   formatR_1.1      grid_3.1.3       gtable_0.1.2
##  [9] htmltools_0.2.6  knitr_1.9        labeling_0.3     lazyeval_0.1.10
## [13] magrittr_1.5     MASS_7.3-39      munsell_0.4.2    parallel_3.1.3
## [17] plyr_1.8.1       proto_0.3-10     Rcpp_0.11.5      reshape2_1.4.1
## [21] rmarkdown_0.5.1  scales_0.2.4     slam_0.1-32      tcltk_3.1.3
## [25] tools_3.1.3      yaml_2.1.13
```