2020 Democratic Debate - Topic Modeling

Wesley Gardiner

2020-05-01

Introduction

Political debates are a big part of the presidential races in the United States. They allow candidates to talk about important issues together as well as share important ideas; this allows for some neat and relevant textual analysis.

While I was looking at datasets for this final project I stumbled upon an interesting data set on Kaggle that were the 2020 Democratic Debate transcripts. After recently learning about topic modeling I decided to choose that data set and apply my new topic modeling skills to answer some questions I had:

- What topics were most common among the candidates?
- Who talked the most?
- What words came up a lot for each candidate?

Some important things to note:

- For the purposes of this final project, only the code for 1 candidate through my report will be shown due to the computational power necessary to carry out one candidate (topic modeling). However, if one wanted to look at any other candidate, they would only need to change one variable `candidate_name` in the Rmd file.
- Another important thing to mention, is that only candidates that stayed until the end of the debate series are shown and analyzed.

Getting the Data

Full link: https://www.kaggle.com/brandenciranni/democratic-debate-transcripts-2020

Importing the Data

Here are some packages I used to do my inital cleaning

```r
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------------------------------------------------
```

```
## v ggplot2 3.3.0     v purrr   0.3.3
## v tibble  3.0.1     v dplyr   0.8.4
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
## -- Conflicts --------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(tidytext)
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:dplyr':
##
##     intersect, setdiff, union

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
library(stringr)
library(ggplot2)
library(tidyr)
library(citr)
```

Read the data into R.

```
#Reads the data in from csv format
debate_data <-
  read.csv(
    here::here("data", "raw", "debate_transcripts_v3_2020-02-26.csv"),
    stringsAsFactors = FALSE,
    encoding = "UTF-8"
  )

#I perfer to use tibbles :)
debate_data <- tibble(debate_data)
```

Specifying `stringAsFactors = FALSE` and `encoding = "UTF-8"` due to conflict between the characters encoding from the original data set.

A peek into the data:

```
#Structure of the data
str(debate_data)
```

```
## tibble [5,911 x 6] (S3: tbl_df/tbl/data.frame)
##  $ date                : chr [1:5911] "02-25-2020" "02-25-2020" "02-25-2020" "02-25-2020" ...
##  $ debate_name         : chr [1:5911] "South Carolina Democratic Debate Transcript: February 25 Demo
##  $ debate_section      : chr [1:5911] "Part 1: South Carolina Democratic Debate Transcript" "Part 1
##  $ speaker             : chr [1:5911] "Norah O<U+0092>Donnell" "Gayle King" "Norah O<U+0092>Donnell
##  $ speech              : chr [1:5911] "Good evening and welcome, the Democratic presidential primary
##  $ speaking_time_seconds: num [1:5911] 8 22 14 10 31 59 5 5 19 27 ...
```

```
#Names of the columns
colnames(debate_data)
```

```
## [1] "date"                   "debate_name"            "debate_section"
## [4] "speaker"                "speech"                 "speaking_time_seconds"
```

```
#This takes a look at the dimensions of our data in a clear format
paste("Our data has",nrow(debate_data),"rows and", ncol(debate_data),"columns")
```

```
## [1] "Our data has 5911 rows and 6 columns"
```

Speaking Time

One of the questions asked was: Who talked the most?

Since the analyses are of candidates that stuck through-out the debates (last one being the South Carolina Debate on Feb. 25) a list of them must be provided.

```
#This will filter out only the last debate items
list_of_speakers_last <-
  debate_data %>%
  filter(debate_name == "South Carolina Democratic Debate Transcript: February 25 Democratic Debate")

#This gives the names of the speakers in the last debate
unique(list_of_speakers_last$speaker)
```

```
##  [1] "Norah O<U+0092>Donnell" "Gayle King"            "Bernie Sanders"
##  [4] "Michael Bloomberg"    "Pete Buttigieg"       "Elizabeth Warren"
##  [7] "Tom Steyer"           "Joe Biden"            "Amy Klobuchar"
## [10] "Bill Whitaker"        "Major Garrett"        "Speaker 1"
## [13] "Margaret Brennan"
```

The speakers can be seen above.

Creating a list of candidates.

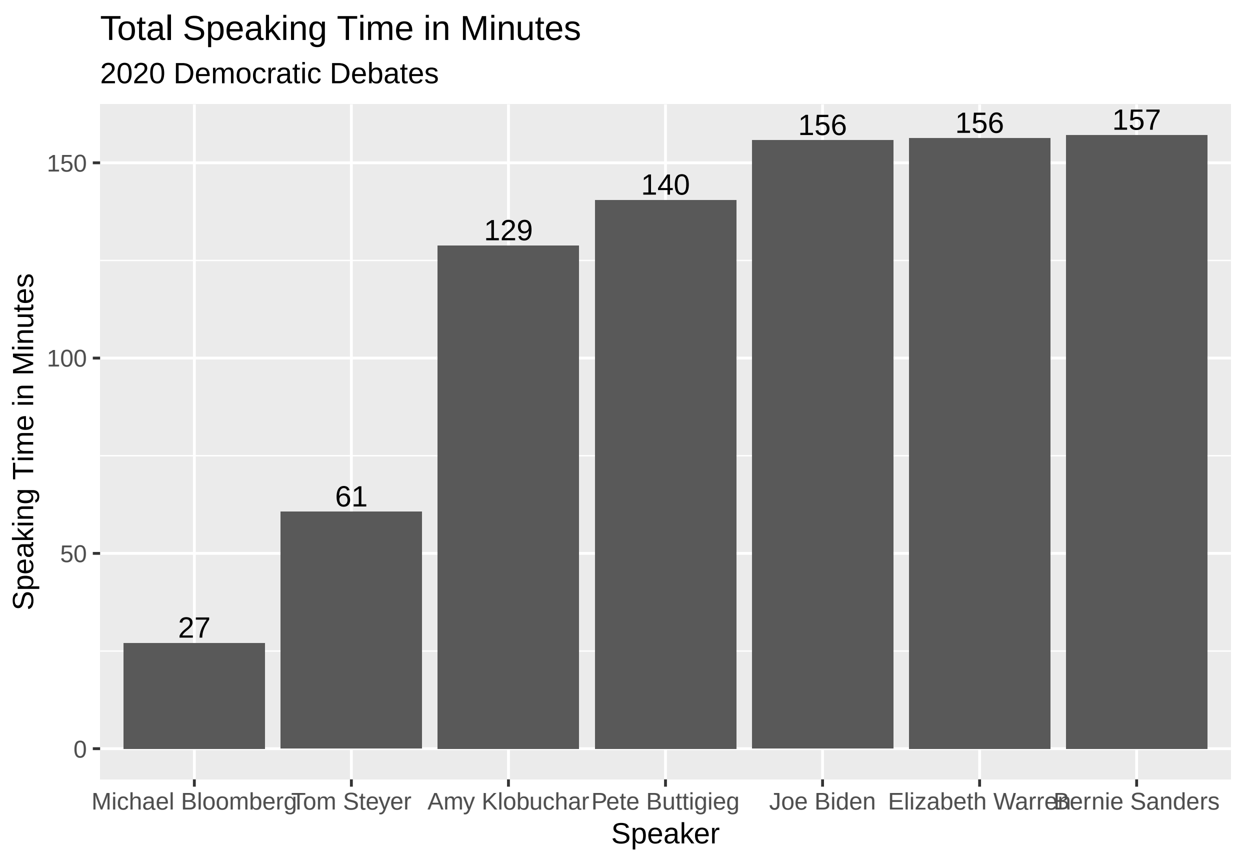
```
#This is a list of the candidates of the last debate
list_of_candidates_last <- c("Bernie Sanders","Joe Biden","Elizabeth Warren","Michael Bloomberg","Pete 
```

Some of the speaking times were 0, changed them to 1 second

```
#There were some speaking times of 0 (which doesn't seem to make sense) so I replaced them with 1 secon
debate_data <-
  debate_data %>%
  mutate(speaking_time_seconds = ifelse(speaking_time_seconds == 0, 1, speaking_time_seconds))
```

Now one can start making a graph of the data to visualize it.

```r
#Here I am going to graph the speaking time in minutes for our candidates
speaking_time_graph <-
  debate_data %>%
  tidyr::drop_na() %>% #Drops NA
  group_by(speaker) %>% # I have to tell it to group by speaker because there are multiple rows of own
  filter(speaker %in% list_of_candidates_last) %>%
  summarise(total_speaking_time_minutes = sum(speaking_time_seconds)/60) %>%  #Creates our minutes colu
  mutate(speaker = fct_reorder(speaker, total_speaking_time_minutes)) %>% #reoders for clarity
  ggplot(aes(speaker, total_speaking_time_minutes)) +
  geom_col()

#Cleans it up a little bit and adds labels
speaking_time_graph +
  labs(
    title = "Total Speaking Time in Minutes",
    subtitle = "2020 Democratic Debates",
    x = "Speaker",
    y = "Speaking Time in Minutes"
  ) +
  geom_text(aes(label=round(total_speaking_time_minutes)), position=position_dodge(width=0.9), vjust=-0
```



We can see here that Sanders had the most speaking time (157 minutes) and Bloomberg had the least (27 minutes) and that makes sense because Bloomberg didn't have start his campaign until later.

Tokenization

We must first tokenize (split into individual words) in order to perform textual analysis. This can be done using the `tidytext` package.

```r
library(tidytext)
```

From here on only one candidate will be shown; however, changing the name will result in different analyses.

```r
#I'm choosing Bernie Sanders because he has the most data.
candidate_name <- "Bernie Sanders"
```

Filter the transcripts for `candidate_name`.

```r
candidate_transcripts <-
  debate_data %>%
  filter(speaker == candidate_name) %>%
  mutate(document = (1:nrow(.))) #Add's a document column
```

Splitting the text into documents is an important step in textual analysis. Each time the candidates speak are chosen as the document type.

Stop Words

Stopwords must be removed from the text (words that carry little information like: a, we, I, me). (University of Liechtenstein et al. 2016, 10)

The tidy text packages has commonly used stop words; however, some custom stopwords must be implemented.

```r
custom_stop_words <- tibble::tribble(
      ~word,        ~lexicon,
      "america",    "custom",
      "american",   "custom",
      "americans",  "custom",
      "people",     "custom",
      "country",    "custom",
      "bring",      "custom",
      "'",          "custom", #The reason I added this one is because of the UTF-8 coding
      "don",        "custom",
      "ve",         "custom",
      "crosstalk",  "custom",
      "ain",        "custom",
      "ll",         "custom",
      "didn",       "custom",
      "president",  "custom",
      "donald",     "custom",
      "time",       "custom",
      "tonight",    "custom",
    )

#Adds my custom stopwords with the already made stopwords
stop_words2 <- stop_words %>% #stop_words comes from the tidytext package
  bind_rows(custom_stop_words)
```

Now tokenize the data.

```r
#Unnested tokens data
candidate_token <-
  candidate_transcripts %>%
  unnest_tokens(word, speech, token = "words") %>%
  anti_join(stop_words2) %>%  #Stop words
  arrange(word)
```

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

Remove any digits (dollar amounts, statistics, etc).

```r
candidate_token <-
  candidate_token %>%
  filter(!grepl("[[:digit:]]", word)) #removes any digits
```

Word Cloud

One nice visualization is a word cloud when it comes to textual data. They provide a nice intuitive way of looking at words that show up a lot.

Imposing thresholds allow for focus in wordclouds. As an arbitrary measure, words that show up more than 7 times are shown on the word cloud.

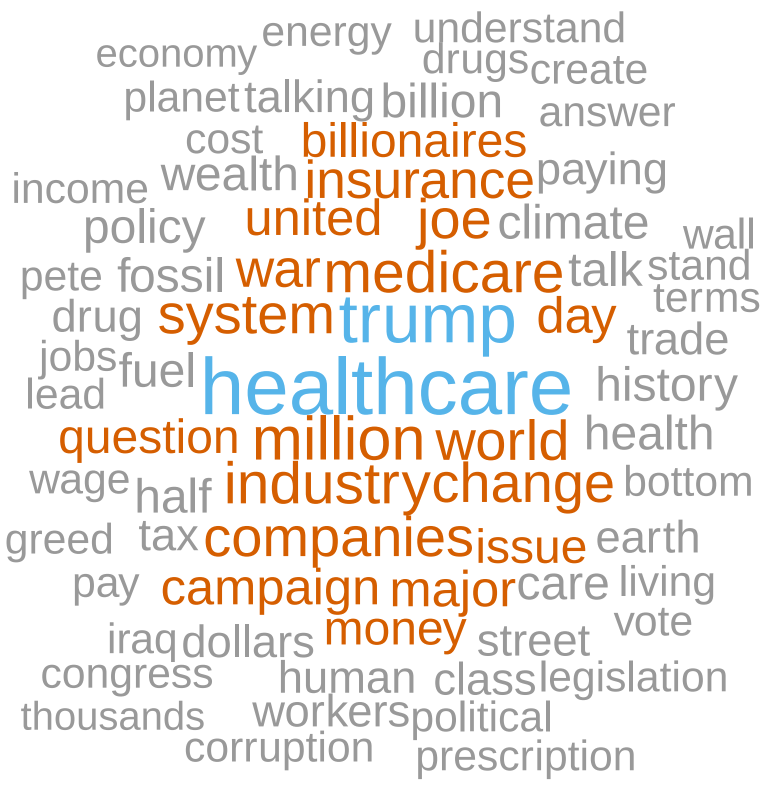
```r
#Creating a word list with words being mentioned > 7 times
candidate_word <- candidate_token %>%
  count(word) %>%
  filter(n > 7)
```

The `wordcloud` package will be used for producing wordclouds.

```r
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

```r
#Color-blind friendly palette
cbPalette <- c("#999999","#D55E00", "#56B4E9")

candidate_cloud <- wordcloud(words = candidate_word$word,
                        freq = candidate_word$n,
                        color = cbPalette,
                        random.order=FALSE,
                        rot.per=0,
                        scale=c(2.5,1),
                        max.words = 70) #Only shows 70 words
```



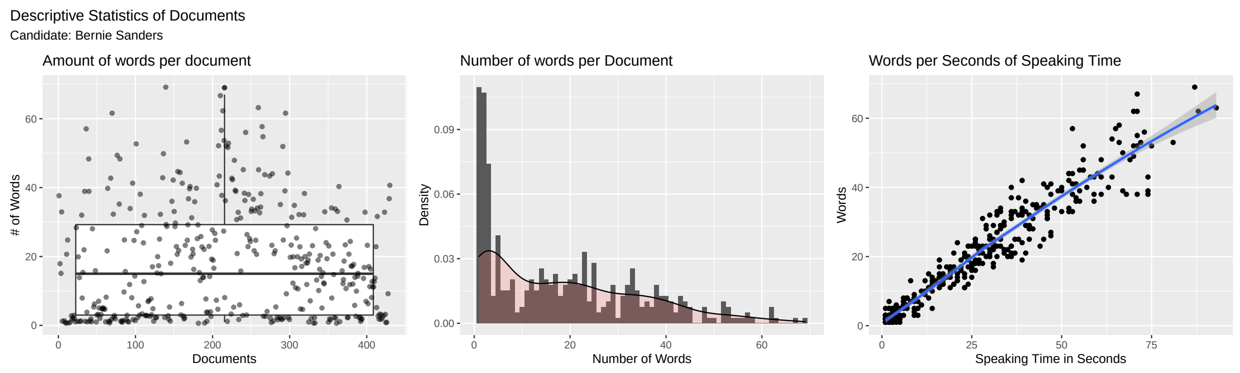
Descriptive Statistics

Before conducting topic modeling, its important to look at descriptive statistics of the data beforehand. Two questions that can lead to fruitful topic modeling are:

- How many tokens (or words) are there?
- What is the distribution of tokens per document?

```r
#gets the count of words per document
words_per_document <-
  candidate_token %>%
  count(document, speaking_time_seconds)

#Creates a dot plot of words by speaking time
words_per_time_plot <-
  ggplot(words_per_document, aes(x = speaking_time_seconds, y = n)) +
  geom_point() +
  geom_smooth() +
  labs(x = "Speaking Time in Seconds",
       y = "Words",
       title = "Words per Seconds of Speaking Time")

#Creates a distribution plot of the number of words per document
distribution_of_words <- ggplot(words_per_document, aes(x = n)) +
  geom_histogram(aes(y = ..density..),       # Histogram with density instead of count on y-axis
                 binwidth = 1) +
  geom_density(alpha = .2, fill = "#FF6666") +
  labs(x = "Number of Words",
       y = "Density",
       title = "Number of words per Document")

# Creates a boxplot with the documents and words
boxplot_amt_words <-
  ggplot(words_per_document, aes(x = document, y = n)) +
  geom_boxplot() +
  geom_jitter(alpha = .5) +
  labs(x = "Documents",
       y = "# of Words",
       title = "Amount of words per document")

#Loads patchwork
library(patchwork)

#Creates a plot with all three graphs
tri_plot <-
  boxplot_amt_words + distribution_of_words + words_per_time_plot +
  plot_annotation(title = "Descriptive Statistics of Documents",
                  subtitle = paste("Candidate:",candidate_name))
tri_plot
```

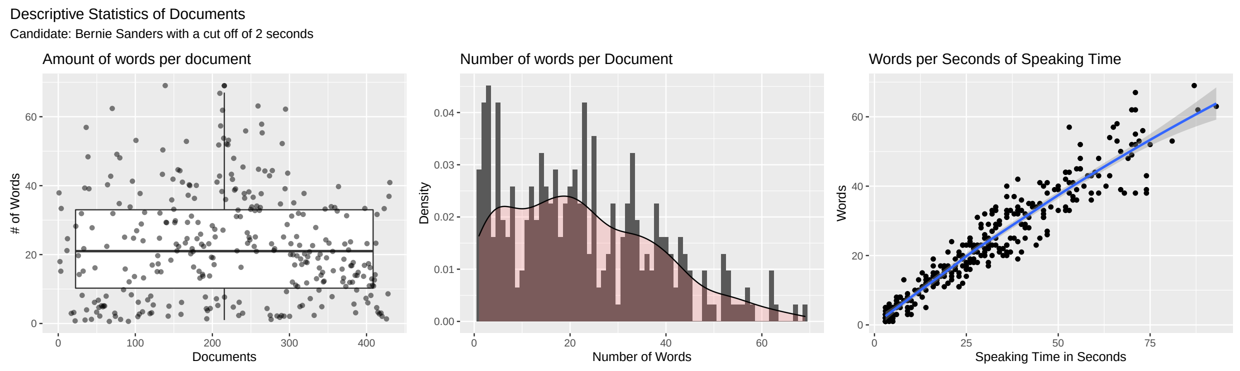
```
## Warning: Continuous x aesthetic -- did you forget aes(group=...)?

## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Looking at the first plot, a large amount of document have a higher density of smaller words. This can be problematic because of the data-hungry nature of topic modeling. It disrupt not only the creation of the model but also the diagnostics of determining the correct number of topics. This is why imposing a threshhold of a `speaking_time_seconds` of more than 2 can be beneficial.s

```
cut_off_time <- 2

candidate_token <-
  candidate_token %>%
  filter(speaking_time_seconds > cut_off_time)
```

```
## Warning: Continuous x aesthetic -- did you forget aes(group=...)?

## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



We can see the change in distribution when enacting the cut off time.

Topic Modeling

The fun part!

Topic modeling is a unsuperived machine learning technique (meaning that it looks at nonlabeled data.) When it comes to topic modeling we need to give the algorithm a number for the number of topics we expect to find. The fun part is trying to figure out what number of topics are in the data. There are many diagnostics that can be used to determine the optimal number of topics and for my model I am going to be showcasing 4: "held-out likelihood","semantic coherence","residuals", and "lower-bound" metrics.("Training, Evaluating, and Interpreting Topic Models | Julia Silge," n.d.; Wallach et al. 2009)

The `stm` package is a great package for doing topic modeling.(Roberts, Stewart, and Tingley 2014; "Training, Evaluating, and Interpreting Topic Models | Julia Silge," n.d.)

```r
library(stm)
```

```
## stm v1.3.5 successfully loaded. See ?stm for help.
##  Papers, resources, and other materials at structuraltopicmodel.com
```

One way we can find the optimal number of topics is by making a model for each number of topics (2-20) and graphing it. We can then look at the estimations.

We need to create something called a "Document-Frequency Matrix."("Training, Evaluating, and Interpreting Topic Models | Julia Silge," n.d.) Basically this is a way of gathering our data in a format that shows the frequency of each word in a collection of documents. It looks kind of like this:

Document	Word1	Word2	Word3	Word 4
1	1	0	0	1
2	0	1	1	1
3	1	0	1	1

```r
#Creating dfm
candidate_dfm <- candidate_token %>%
  count(document,word,sort = TRUE) %>%
  tidytext::cast_dfm(document,word,n)
```

We also need a sparse matrix ("Training, Evaluating, and Interpreting Topic Models | Julia Silge," n.d.)

```
#Creates a cast_spares
candidate_sparse <- candidate_token %>%
  count(document, word) %>%
  tidytext::cast_sparse(document, word, n)
```

This way of evaluating an optimal amount of topics is comupationally heavy. That's why I'm going to use the `furrr` package for parallel computing.

```
library(furrr)
```

```
## Loading required package: future
```

```
library(ggthemes) #For some themes for graphs
```

This tells the computer to plan for a multiprocess form of the package

```
future::plan(multiprocess)
```

Here we are making a dataframe that will have all of our models (2-20). I chose the `init.type = "Spectral"` because that is what is suggested from the stm authors. (Roberts, Stewart, and Tingley 2014, 10)

```
set.seed(278)

many_models <-
  data_frame(K = c(2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20)) %>%
  mutate(topic_model = future_map(K, ~ stm(
    candidate_sparse, K = .,
    verbose = FALSE,
    init.type = "Spectral"
  )))
```

```
## Warning: `data_frame()` is deprecated as of tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
## Warning in stm(candidate_sparse, K = ., verbose = FALSE, init.type =
## "Spectral"): K=2 is equivalent to a unidimensional scaling model which you may
## prefer.
```

Here we make a selection of what documents we want to hold out from the model when evaluating it. ("Training, Evaluating, and Interpreting Topic Models | Julia Silge," n.d.)

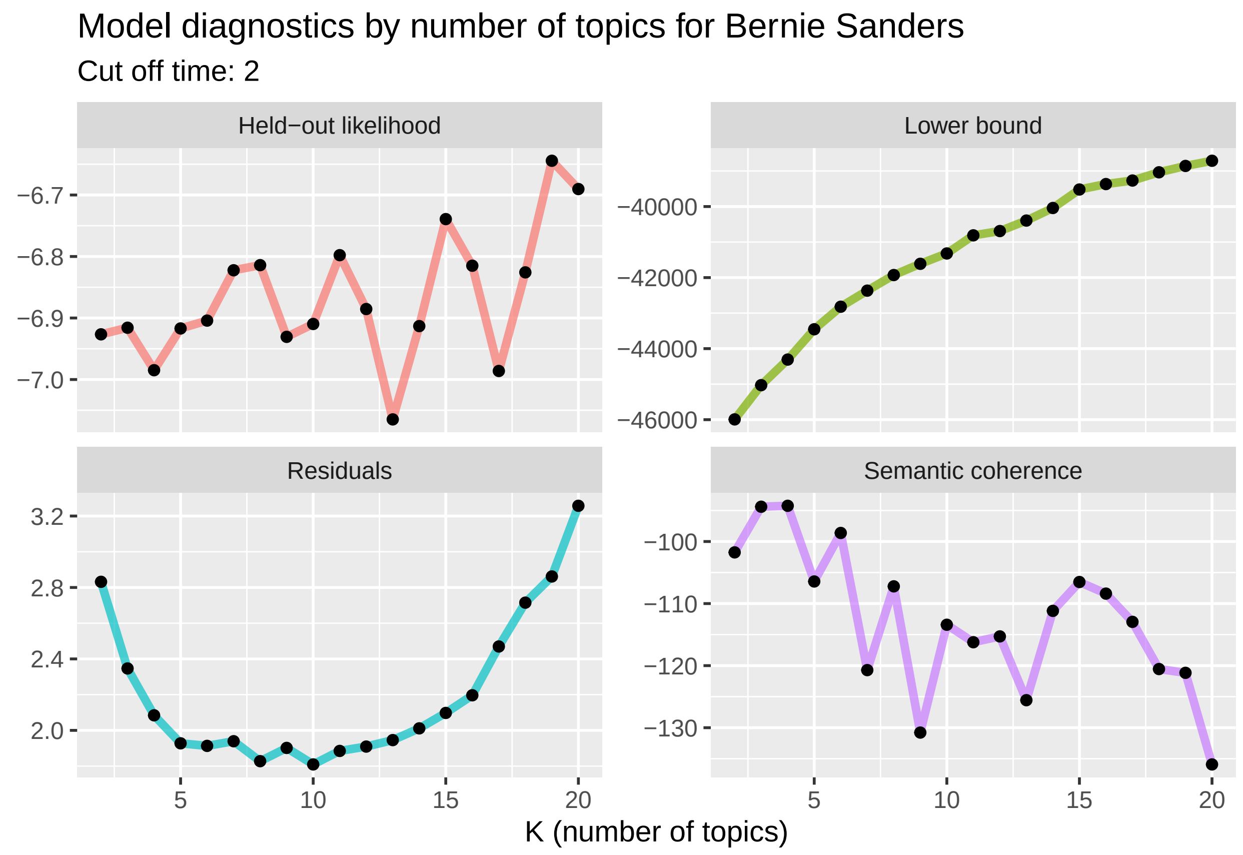
```r
#Makes a heldout
heldout <- make.heldout(candidate_sparse)
#heldout <- make.heldout(candidate_sparse, N = (.05*nrow(candidate_transcripts)), proportion = .25)
```

Now we can create our data frame in which will output the metrics.

```r
#Creates a df of our metrics *from Julia Silge's Blog*
k_result <- many_models %>%
  mutate(
    exclusivity = map(topic_model, exclusivity),
    semantic_coherence = map(topic_model, semanticCoherence, candidate_sparse),
    eval_heldout = map(topic_model, eval.heldout, heldout$missing),
    residual = map(topic_model, checkResiduals, candidate_sparse),
    bound =  map_dbl(topic_model, function(x)
      max(x$convergence$bound)),
    lfact = map_dbl(topic_model, function(x)
      lfactorial(x$settings$dim$K)),
    lbound = bound + lfact,
    iterations = map_dbl(topic_model, function(x)
      length(x$convergence$bound))
  )
```

Now we can visualize the output.

```r
#Cleans the data for the graph
clustergraph_data <- k_result %>%
  transmute(
    K,
    `Lower bound` = lbound,
    Residuals = map_dbl(residual, "dispersion"),
    `Semantic coherence` = map_dbl(semantic_coherence, mean),
    `Held-out likelihood` = map_dbl(eval_heldout, "expected.heldout")
  ) %>%
  gather(Metric, Value,-K)

#Graphs the metrics.
clustergraph <-
  clustergraph_data %>%
  ggplot(aes(K, Value, color = Metric)) +
  geom_line(size = 1.5,
            alpha = 0.7,
            show.legend = FALSE) +
  geom_point(alpha = 1, color = "Black") +
  facet_wrap( ~ Metric, scales = "free_y") +
  labs(
    x = "K (number of topics)",
    y = NULL,
    title = paste("Model diagnostics by number of topics for",candidate_name),
    subtitle = paste("Cut off time:",cut_off_time))
clustergraph
```



As we can see from our diagnostics, the held-out-likelihood peaks at 11 topics. That paried with low residuals leads me to think that 15 is the best number of topics.

We extract the topic with 15 models

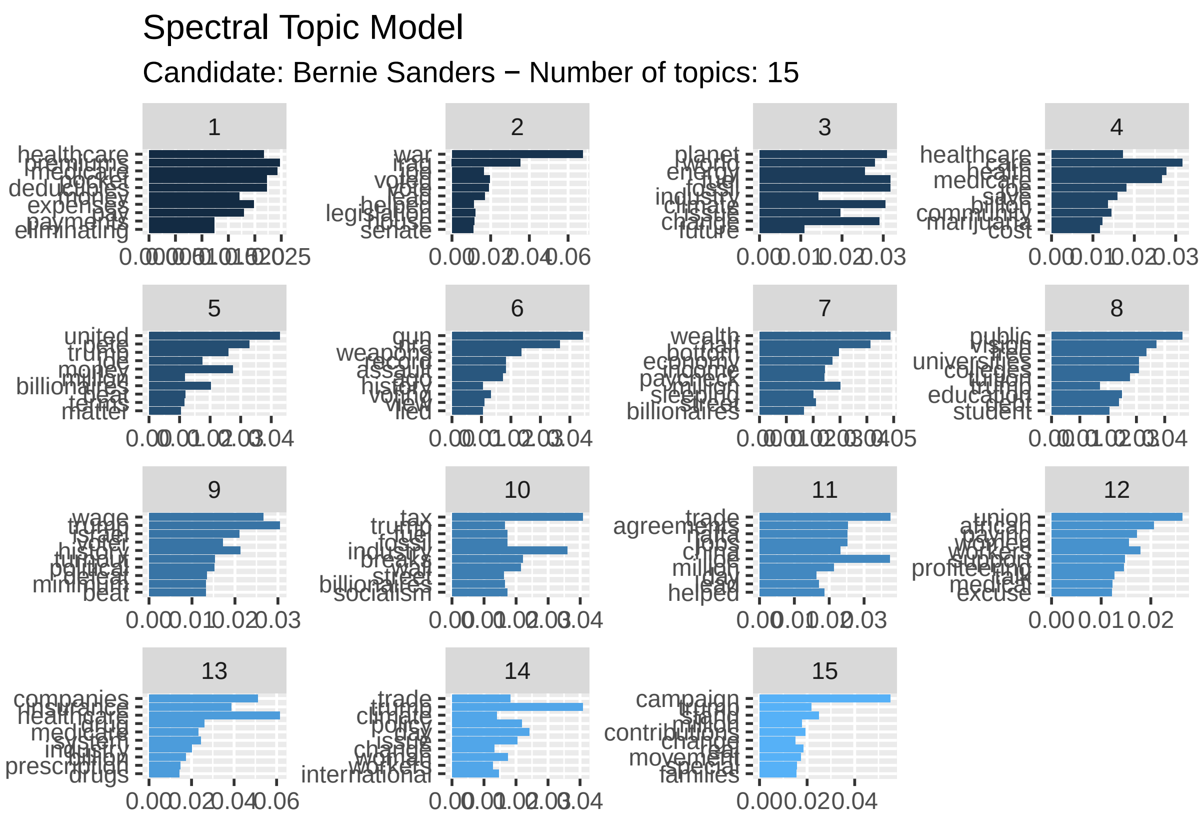
```
number_of_clusters = 15

 topic_model <- k_result %>%
  filter(K == number_of_clusters) %>%
  pull(topic_model) %>%
  .[[1]]
```

We are intrested in beta because that refers to topic-word density. ("LDA Alpha and Beta Parameters - the Intuition - the Thought Vector Blog - Blog Vector," n.d.)

```
#tidy format for the beta measurment
td_beta <- tidy(topic_model)

# visualize the topics
topic_graph <-
  td_beta %>%
  group_by(topic) %>%
  top_n(10) %>%
  ungroup %>%
  mutate(term = reorder(term, beta)) %>% #ordering it by their beta rank
  ggplot(aes(term, beta, fill = topic)) + #plotting it
  geom_col(show.legend = FALSE) +
  facet_wrap( ~ topic, scales = "free") + #by topic
  coord_flip() +
  labs(title = "Spectral Topic Model",
       subtitle = paste("Candidate:",candidate_name,"-","Number of topics:",number_of_clusters),
       x ='',
       y ='')
```

```
## Selecting by beta
```

```
topic_graph
```



Analysis

Looking at the different topics per candidate its interesting to see that one common topic was summed up as "Defeating Trump" across candidates. It also was cool to see how the topics found for each candidate were comprehensive and show words associated around political issues (environment, gun laws, etc). With this type of analysis we can see that debates are filled with key words assocated with political issues; we can also see which candidate use which words when it comes to these issues.

Now I think its important to mention somethings. Topic modeling works best when theres a lot of data. Some candidates had more data than others making topic models easy in some cases and hard in others.

References

“LDA Alpha and Beta Parameters - the Intuition - the Thought Vector Blog - Blog Vector.” n.d. https://www.thoughtvector.io/blog/lda-alpha-and-beta-parameters-the-intuition/.

Roberts, Margaret E, Brandon M Stewart, and Dustin Tingley. 2014. “Stm: R Package for Structural Topic Models.” *Journal of Statistical Software* 10 (2): 1–40.

“Training, Evaluating, and Interpreting Topic Models | Julia Silge.” n.d. https://juliasilge.com/blog/evaluating-stm/.

University of Liechtenstein, Stefan Debortoli, Oliver Müller, IT University of Copenhagen, Iris Junglas, Florida State University, Jan vom Brocke, and University of Liechtenstein. 2016. “Text Mining for Information Systems Researchers: An Annotated Topic Modeling Tutorial.” *Communications of the Association for Information Systems* 39: 110–35. https://doi.org/10.17705/1CAIS.03907.

Wallach, Hanna M., Iain Murray, Ruslan Salakhutdinov, and David Mimno. 2009. “Evaluation Methods for Topic Models.” In *Proceedings of the 26th Annual International Conference on Machine Learning - ICML ’09*, 1–8. Montreal, Quebec, Canada: ACM Press. https://doi.org/10.1145/1553374.1553515.