2020 Democratic Debate - Topic Modeling

Wesley Gardiner

2020-05-01

## Introduction

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, I am only going to show the code to 1 candidate through my report due to the computational power necessary to carry out one candidate (I’m looking at you topic modeling). However, if you wanted to look at any other candidate, I will make it as easy as changing a variable
* Another important thing I should mention, is that I only looked at candidates that stayed until the end of the debate series.

So lets get started!

## Getting the Data

As mentioned previous, I was able to get the data from Kaggle.com

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

library(tidyverse)

## -- Attaching packages ---------------------------------------------------------------------------------- tidyverse 1.3.0 --

## 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 ------------------------------------------------------------------------------------- tidyverse\_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)

If you don’t have these don’t fret! You can download them using these commands

# install.packages("tidyverse")  
# install.packages("tidytext")  
# install.packages("lubridate")  
# install.packages("stringr")  
# install.packages("ggplot2")  
# install.packages("tidyr")

First thing I need to do is read the data into R (Since the data comes in a CSV format, I use the read.csv() function)

#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)

I specify stringAsFactors = FALSE because I don’t want the function to read characters or strings and label them as factors. I also specify an encoding of “UTF-8” because there was some conflict between the characters encoding from the original data set.

Great! Now lets take a peek into our 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 Democratic Debate" "South Carolina Democratic Debate Transcript: February 25 Democratic Debate" "South Carolina Democratic Debate Transcript: February 25 Democratic Debate" "South Carolina Democratic Debate Transcript: February 25 Democratic Debate" ...  
## $ debate\_section : chr [1:5911] "Part 1: South Carolina Democratic Debate Transcript" "Part 1: South Carolina Democratic Debate Transcript" "Part 1: South Carolina Democratic Debate Transcript" "Part 1: South Carolina Democratic Debate Transcript" ...  
## $ speaker : chr [1:5911] "Norah O<U+0092>Donnell" "Gayle King" "Norah O<U+0092>Donnell" "Gayle King" ...  
## $ speech : chr [1:5911] "Good evening and welcome, the Democratic presidential primary here in South Carolina. The first primary in the "| \_\_truncated\_\_ "And Super Tuesday is just a week away and this is the biggest primary day of the year as voters in 14 states ca"| \_\_truncated\_\_ "And CBS News is proud to bring you this debate along with our co-sponsors. They are the Democratic National Com"| \_\_truncated\_\_ "And we are partnering tonight also with Twitter. So you at home can participate in this debate. How do you do t"| \_\_truncated\_\_ ...  
## $ 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 my questions was: Who talked the most?

Since I’m only analyzing candidates that stuck through-out the debates (last one being the South Carolina Debate on Feb. 25) I need to figure out who they are.

#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"

As we can see from the output above, those are the speakers that stuck throughout.

Now I can start looking at the speaking times.

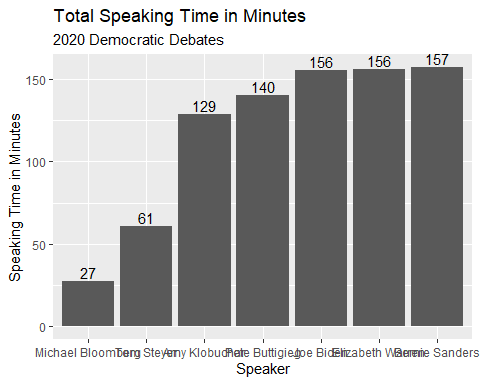
#This is a list of the candidates of the last debate  
list\_of\_candidates\_last <- c("Bernie Sanders","Joe Biden","Elizabeth Warren","Michael Bloomberg","Pete Buttigieg","Tom Steyer","Amy Klobuchar")

I noticed that some of the speaking times were 0. Now, that doesn’t make much sense to me so I changed every time the speaking time was 0 seconds to 1 second

#There were some speaking times of 0 (which doesn't seem to make sense) so I replaced them with 1 second.  
debate\_data <-  
 debate\_data %>%   
 mutate(speaking\_time\_seconds = ifelse(speaking\_time\_seconds == 0, 1, speaking\_time\_seconds))

Now I can get started with making a graph of the data to visualize it.

#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 speaker  
 filter(speaker %in% list\_of\_candidates\_last) %>%   
 summarise(total\_speaking\_time\_minutes = sum(speaking\_time\_seconds)/60) %>% #Creates our minutes column  
 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=-0.25)



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

When it comes to textual data we need to have our data in a specific format. We want to “tokenize” each time they speak so we can run algorithms on it. To “tokenize” something (in textual analysis) means, to break it up into every word. Thanks to the tidytext package, thats easy!

library(tidytext)

This is where I will be showing only 1 candidate as an example; however, I’ll make a variable called candidate\_name that you can freely change and it will change the analysis.

#I'm choosing Bernie Sanders because he has the most data.  
candidate\_name <- "Bernie Sanders"

We need to get all the transcripts for our candidate\_name.

candidate\_transcripts <-  
 debate\_data %>%   
 filter(speaker == candidate\_name) %>%   
 mutate(document = (1:nrow(.))) #Add's a document column

In textual analysis, every time there is a object of text (sentence, paragraph, chapter, book) those are called documents and we select how we want to classify a document during our cleaning process. I chose to use each time a candidate spoke as a document (hence the document column).

### Stop Words

We need to remove stopwords 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, but I add some of my own that I found came up in all candidates.

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 our data.

#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 anytime there was a digit.

candidate\_token <-   
 candidate\_token %>%  
 filter(!grepl("[[:digit:]]", word)) #removes any digits

With our data clean, we can get to the fun part.

## Word Cloud

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

We want to look at words that show up the most so we put a threshhold of words that show up more than 7 times (7 is an arbitrary number; when picking a number it depends on your data)

#Creating a word list with words being mentioned > 7 times  
candidate\_word <- candidate\_token %>%  
 count(word) %>%  
 filter(n > 7)

We’ll use the wordcloud package for our wordcloud.

library(wordcloud)

## Loading required package: RColorBrewer

#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

## Warning in wordcloud(words = candidate\_word$word, freq = candidate\_word$n, :  
## political could not be fit on page. It will not be plotted.

## Warning in wordcloud(words = candidate\_word$word, freq = candidate\_word$n, :  
## understand could not be fit on page. It will not be plotted.

## Warning in wordcloud(words = candidate\_word$word, freq = candidate\_word$n, :  
## congress could not be fit on page. It will not be plotted.

## Warning in wordcloud(words = candidate\_word$word, freq = candidate\_word$n, :  
## corruption could not be fit on page. It will not be plotted.

## Warning in wordcloud(words = candidate\_word$word, freq = candidate\_word$n, :  
## legislation could not be fit on page. It will not be plotted.

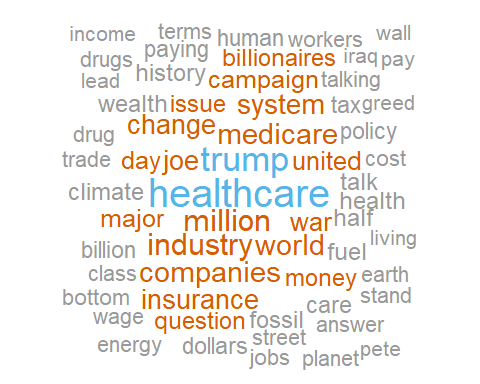
## Warning in wordcloud(words = candidate\_word$word, freq = candidate\_word$n, :  
## prescription could not be fit on page. It will not be plotted.

## Warning in wordcloud(words = candidate\_word$word, freq = candidate\_word$n, :  
## vote could not be fit on page. It will not be plotted.

## Warning in wordcloud(words = candidate\_word$word, freq = candidate\_word$n, :  
## create could not be fit on page. It will not be plotted.

## Warning in wordcloud(words = candidate\_word$word, freq = candidate\_word$n, :  
## bloomberg could not be fit on page. It will not be plotted.

## Warning in wordcloud(words = candidate\_word$word, freq = candidate\_word$n, :  
## community could not be fit on page. It will not be plotted.



## Descriptive Statistics

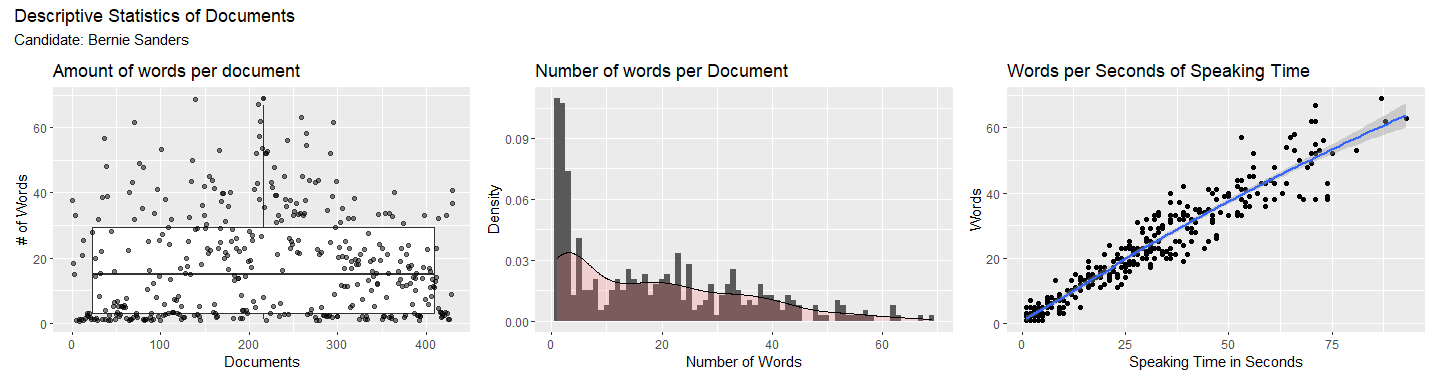
When doing topic modeling, its a good idea to get a better understanding of your data with looking at a couple of things:

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

#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'

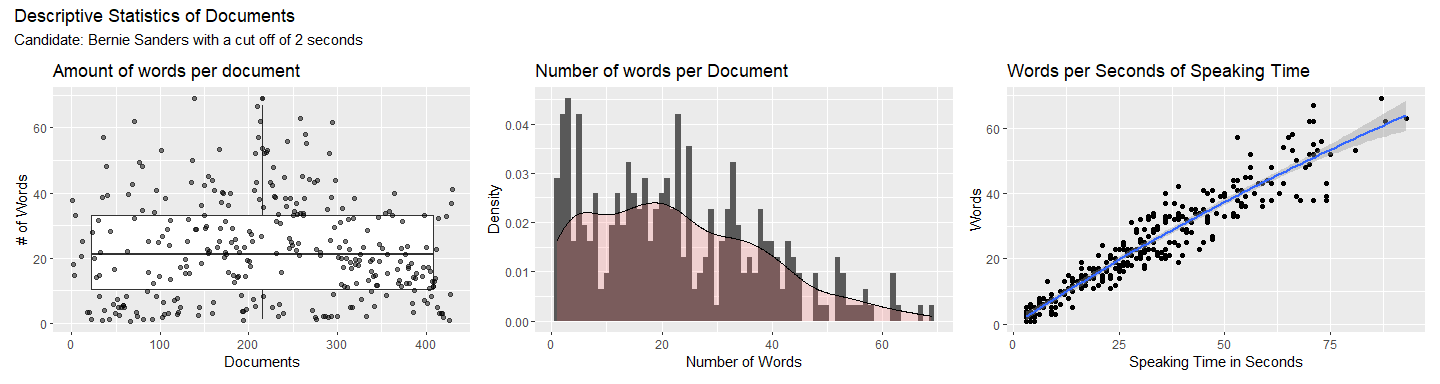


If we see from the first two plots, a lot of the documents have don’t have a lot of words show up a lot. This can be problematic because of the data-hungry nature of topic modeling. It can mess with not only our model but also the diagnostics of determining the correct number of topics (which we’ll see later). This is why I chose to only include document that have a speaking\_time\_seconds of more than 2

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.)

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 |

#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.)

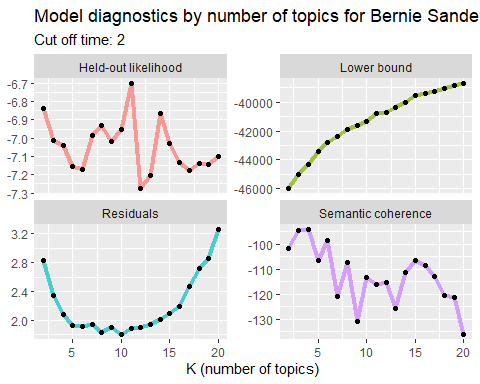
#Makes a heldout  
heldout <- make.heldout(candidate\_sparse, N = (.05\*nrow(candidate\_transcripts)), proportion = .25)

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

#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.

#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 11 is the best number of topics.

We extract the topic with 11 models

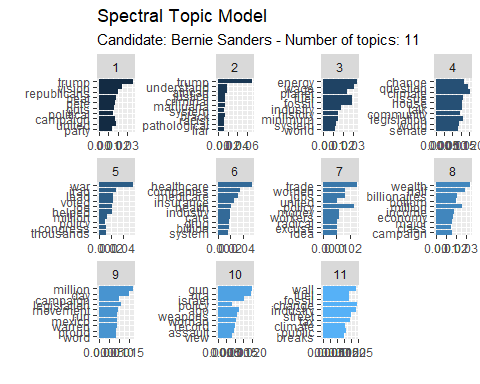
number\_of\_clusters = 11  
  
 topic\_model <- k\_result %>%  
 filter(K == number\_of\_clusters) %>%  
 pull(topic\_model) %>%  
 .[[1]]

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>.