Linguistic Partisanship Analysis of U.S Legislators

Github Repository: https://github.com/STA160/mainProject
Overview Slides: http://bit.ly/2pSWxHz

Randy White
Katherine Olson
Graham Smith

STA 160, Spring 2017

Table of Contents

Abstract

It is immensely important that voters political entities know truly where their constituents lean in terms of their implicit political affiliation, and bias. The intensely polarized nature of our national legislature in terms of voting behavior is well documented at this point[1], but we believe there is value in trying to trace this polarization in the language that legislators actually use. We set to examining this bias through Delta Term Frequency Inverse Document Frequency (DTF-IDF), which looks at the frequency of words that are more common within some subgroup, relative to the group as a whole. Exploring in particular how language varies over time by party, state, and individuals between 2006 and 2017. We discovered that there were indeed large amounts of variation, and even looking at only vocabulary the differences showed splits in political ideology.

Stakeholders

While this paper should be interesting to the general public, we see its value more to professional political scientists to aid in their own analysis. The intensely polarized nature of of Either from examining the cleaned data, running our scripts themselves, or using the TF-IDF scoring as a means of weighting words before performing sentiment analysis.

Data Acquisition

Congressional data was acquired from the "Daily Digest" summaries released as PDFs by congress as part of the official congressional record. We downloaded all available records from May 2017 back to February 2006 via the URL downloader "wget", implemented in Perl. Biographical information was acquired as JSON files through the @unitedstates public

[1] *The Rise of Partisanship and Super-Cooperators in the U.S. House of Representatives*, Amdros+, 2015

GitHub repository, and popularity data was scraped from the Morning Consult polling group's website.

Data Preprocessing

Since our data was acquired in PDF's, our first step was to convert them to .txt files which could be parsed using a command line tool called 'pdftotext'[2]. We then broke up each file by individual speakers, removed obvious errors, and performed some routine data cleaning (removal of stopwords, punctuation, unicode, numbers, etc.). After being linked with publicly available biographical information to determine party affiliation and state, we ran our script for every daily congressional digest back to 2006 and collated them into a single CSV file.

Methods

We primarily used Term Frequency Inverse Document Frequency, a numerical statistic that reflects how important a word is to a document given how frequently it appears in an overall corpus of documents. In our case 'document' is defined as a line of dialogue delivered by a legislator at a single time (usually one to five sentences). While there are multiple ways of calculating this statistic, for **term frequency** $\mathrm{tf}(t,d)$ we chose the simplest method of *raw count* of a term in a document, i.e. the number of times that term t occurs in document d. denoted as $f_{t,d}$

$$\mathrm{tf}(t,d) = f_{t,d}$$

As it seemed unnecessary to utilize more complicated forms of scaling, and all the documents were roughly the same length meaning there was no need to correct for the bias you would find with particularly long documents.

The **inverse document frequency** idf*(t, D)* is a measure of how much information word t provides across all documents D. It is the logarithmically scaled inverse fraction of

[2] poppler.freedesktop.org

the documents that contain the word, obtained by dividing the total number of documents *N* by the number of documents containing the term, and then taking the logarithm of that quotient.

$$\mathrm{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

Out of the several possible weighting schemes for $|\{d \, \varepsilon \, D : t \, \varepsilon \, d\}|$ we chose:

$$\log \frac{N}{n_t} = -\log \frac{n_t}{N}$$

Our justification for which primarily comes from a 2009 paper out of the University of Maryland[3], which used support vector machines to prove its superior accuracy in general situation[4]. TF-IDF is then calculated as

$$\mathrm{tfidf}(t, d, D) = \mathrm{tf}(t, d) \cdot \mathrm{idf}(t, D)$$

A high weight value is reached by high term frequency (in a given document) and a low document frequency of the term in the whole collection of documents; hence, the weights tend to filter out common terms. Since the ratio inside the IDF's log function is always greater than or equal to 1, the value of idf (and TF-IDF) is greater than or equal to 0. As a term appears in more documents, the ratio inside the logarithm approaches 1, bringing the idf and TF-IDF closer to 0.

Delta TF-IDF (DTF-IDF*)* is a classification procedure where we aggregate TF-IDF values and then score them as a form of classification of what is more Republican or Democrat. We then go back to our original documents, sum the classified Republican and Democrat words of an individual, then take the difference of the accumulated vocabulary. Thus, a **more negative score** indicates the collection of words is **more democratic**, and a **more positive score** means they are more **republican**.

[3] Some alternatives are to smooth by adding 1 to the whole term, only look at max frequency for any one term, or a probabilistic method where N is replaced with N - n_t

[4] *Delta TFIDF: An Improved Feature Space for Sentiment Analysis*, Martineau and Finin, 2009

Analysis

The first step in our investigation involved averaging the DTF-IDF scores within a particular slice of the data, and plotting to investigate potential patterns. In fig. 1 below we see that while the legislature was roughly neutral between 2006 and 2013, it took a moderate democratic shift in 2013-2015 before an extreme republican shift in 2016. We can imagine that this is due to the the wild effects of the 2016 election, and democrats being forced to use language that had previously been exclusively republican.

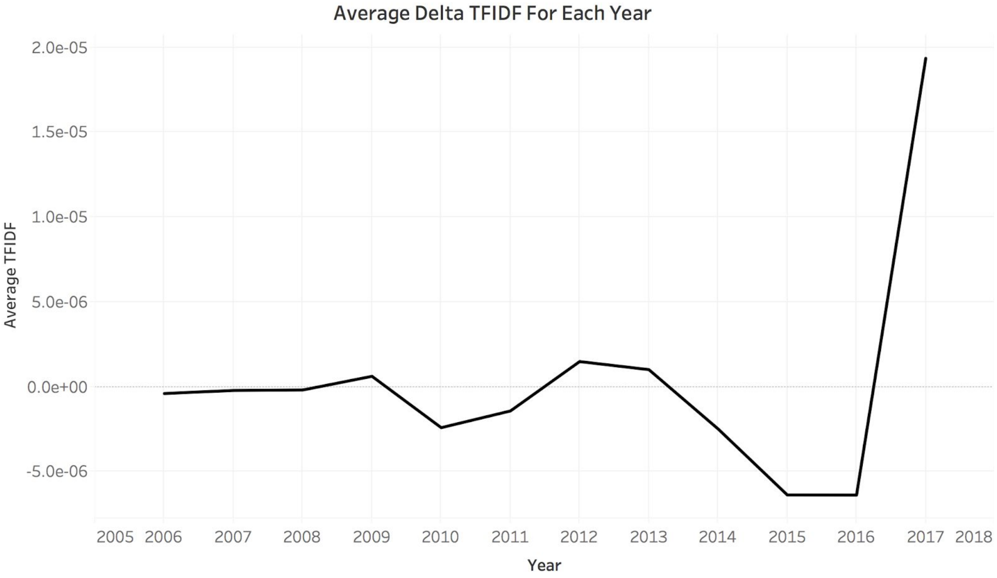


Fig.1: Legislature became dramatically more republican in 2016

While Republicans on average have a DTF-IDF score 4.348e^{-06} (284%) higher than Democrats (fig. 2), They still tend to follow the same pattern.

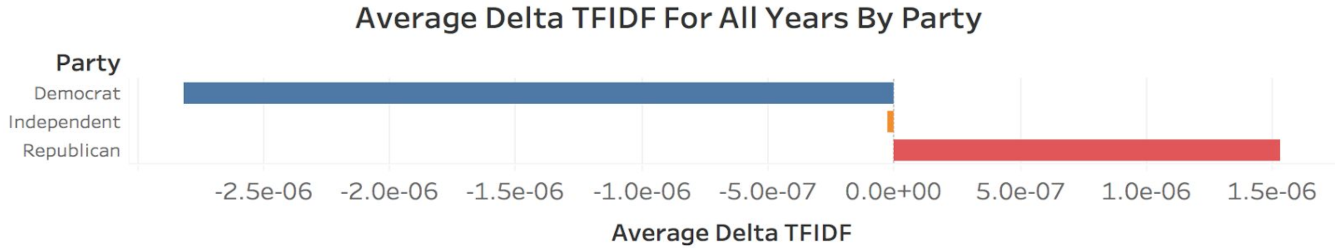


Fig. 2: Average Rep. score is 1.5309e^{-06}, average Dem. score is -2.8174e^{-06}

Looking at the individual parties in fig. 3, we see roughly what we would expect; Both groups tend to shift up and down together as the main topics of discussion move in traditionally liberal or conservative directions.

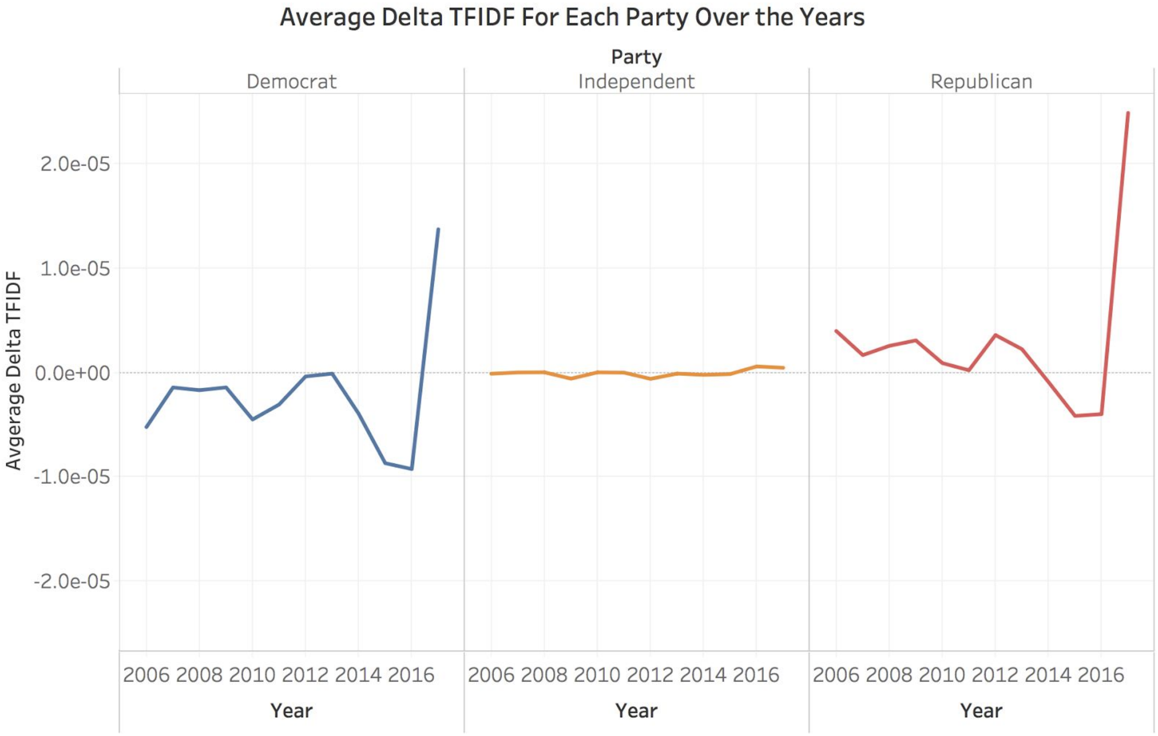
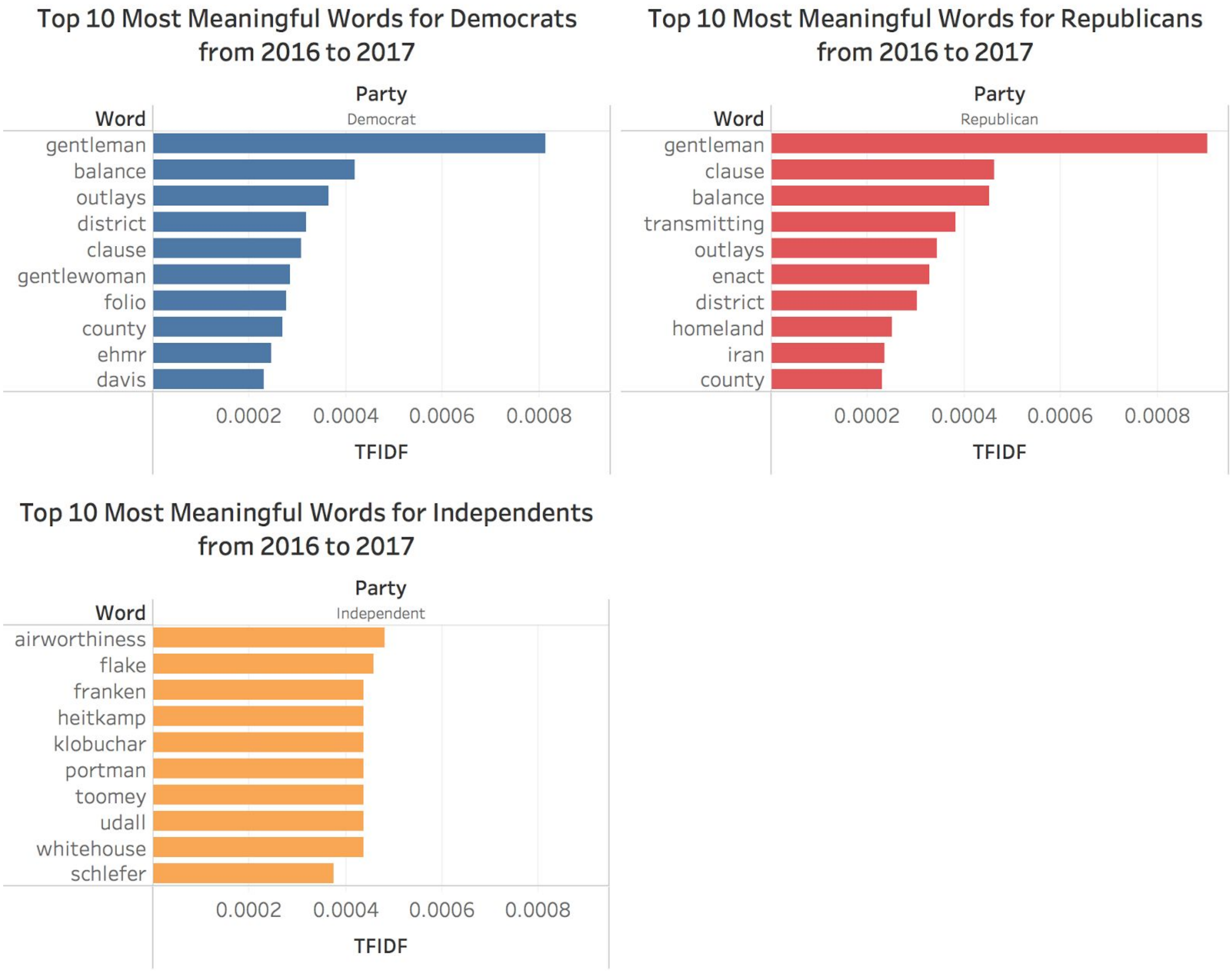


Fig. 3: Parties shift together, Independents follow different topics

An interesting feature is how Independents do not move much at all. While we would expect them to be somewhat in the middle in terms of absolute score, it is surprising that they do not follow the topic trends as the two major parties do. This can possibly be explained by the fact that they are less likely to hold important positions and so are going to have a lot less bureaucratic noise. We can see in Fig. 4 on the following page that while Republicans and Democrats share a lot of uninteresting words in common ('gentleman/woman', 'balance', 'outlays', 'district', etc.) Independents almost exclusively show individual names.



Top 10 Most Meaningful Words for Democrats from 2016 to 2017

Top 10 Most Meaningful Words for Republicans from 2016 to 2017

Top 10 Most Meaningful Words for Independents from 2016 to 2017

Fig. 4: Rep./Dem. have a lot in common, Independent's words are mostly names

To get some context for this data, we can also look at the more stable period of 2006 - 2012 to establish a baseline and see if the same useless words show up.

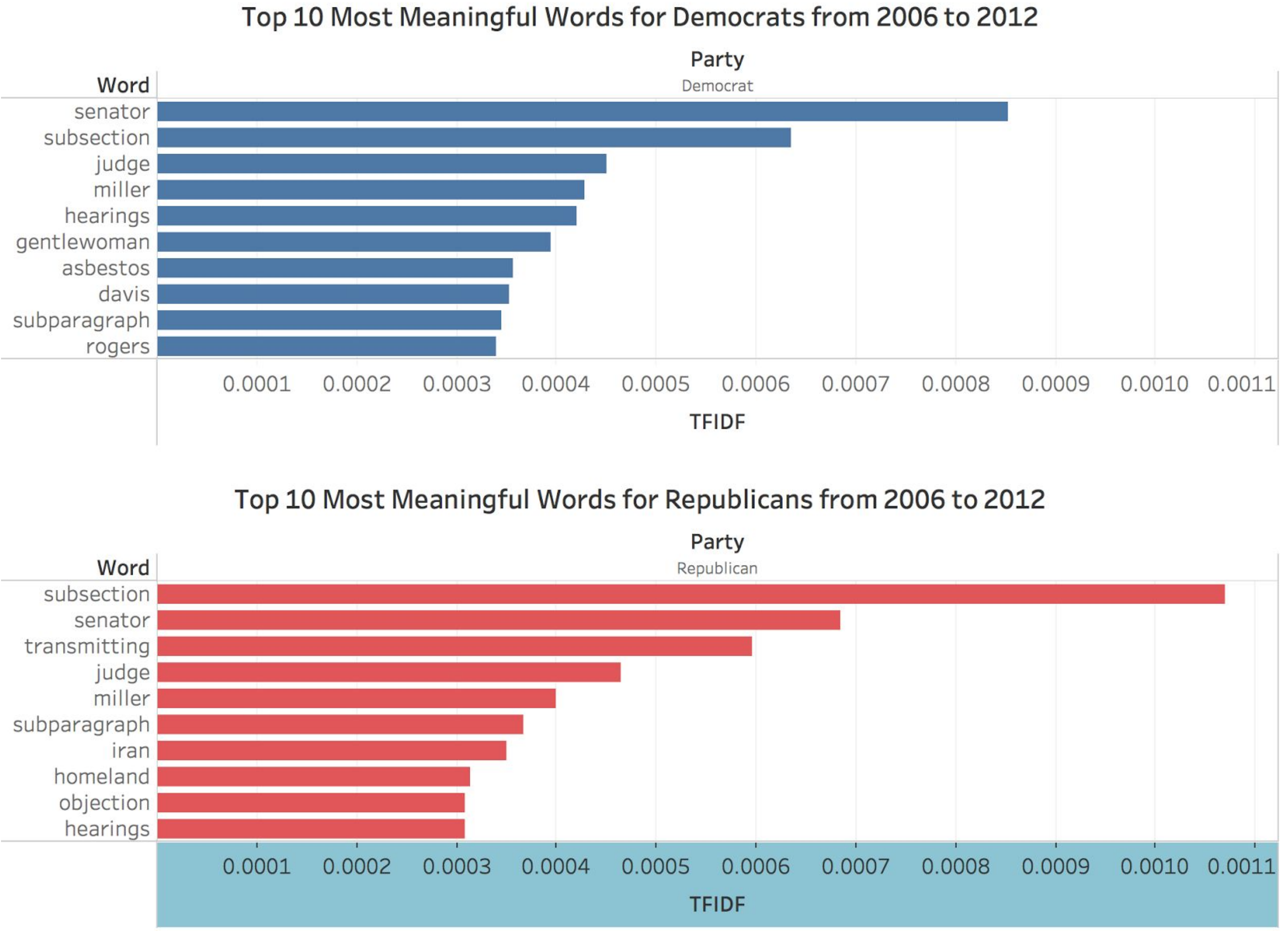


Fig. 5: There is significant word overlap still

There are a few standouts, notably 'Iran', and 'Homeland' for Republicans, showing their focus on security, and 'Asbestos' for Democrats showing a focus on more domestic concerns. However it is hard to read into this too much, and there is clearly a significant amount of noise we have not managed to remove. A massive segment of words remains as non-predictive bureaucratic junk that was difficult to pinpoint.

Looking at the data from a different direction, we plotted DTF-IDF scores by state as well. Primarily to see if the degree of polarization matched up with our expectations as a sanity check.

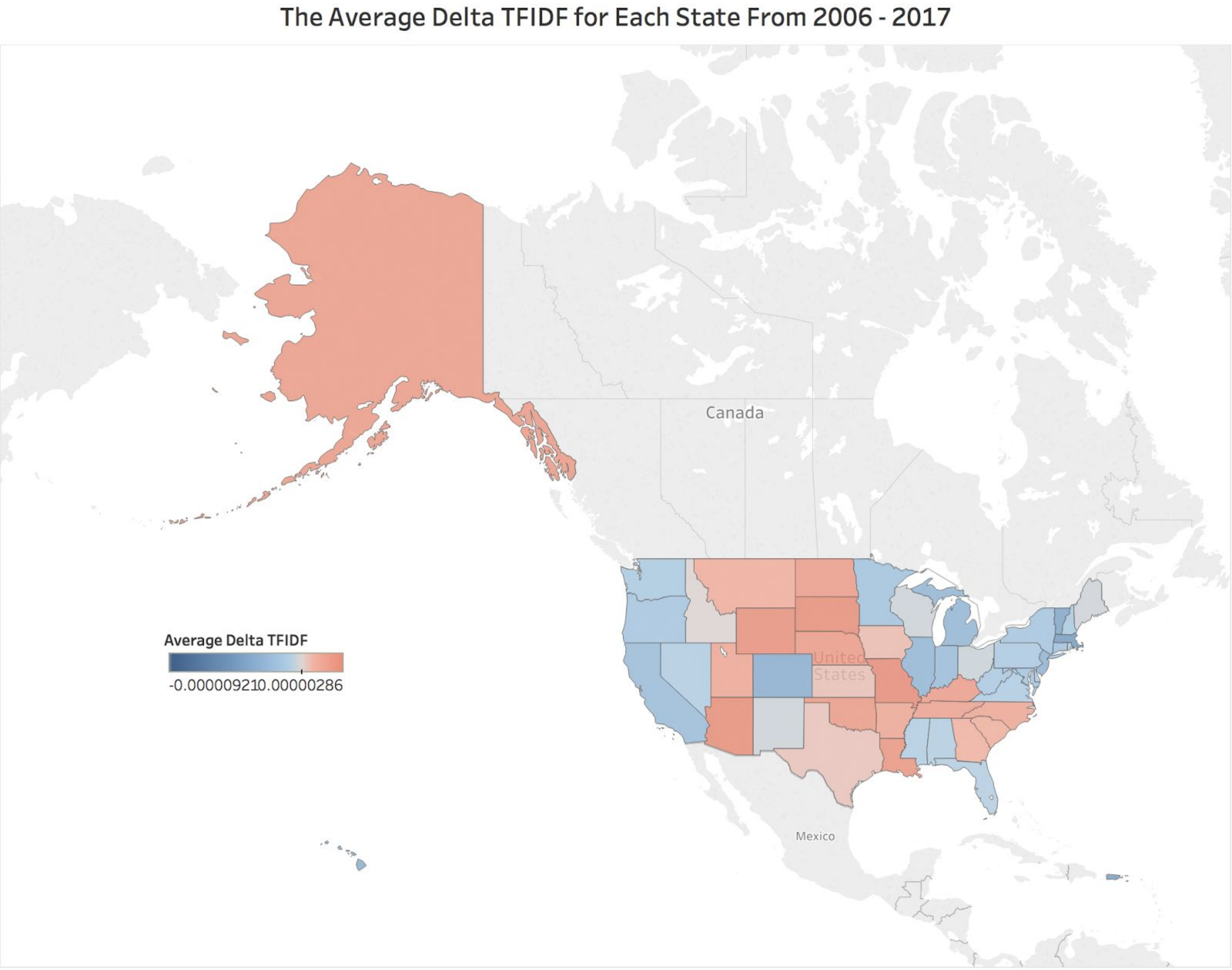

The Average Delta TFIDF for Each State From 2006 - 2017

Fig. 5: TFIDF scores are roughly in line with traditional expectations

This roughly lines up with what we know regarding the political leanings of each state. The standout exceptions seem to all be in traditionally conservative states. Notably Mississippi, Alabama, Florida, and West Virginia.

But, who are the individual legislators at the highest and lowest ends of the spectrum that should, one would imagine, have the least amount of overlap?

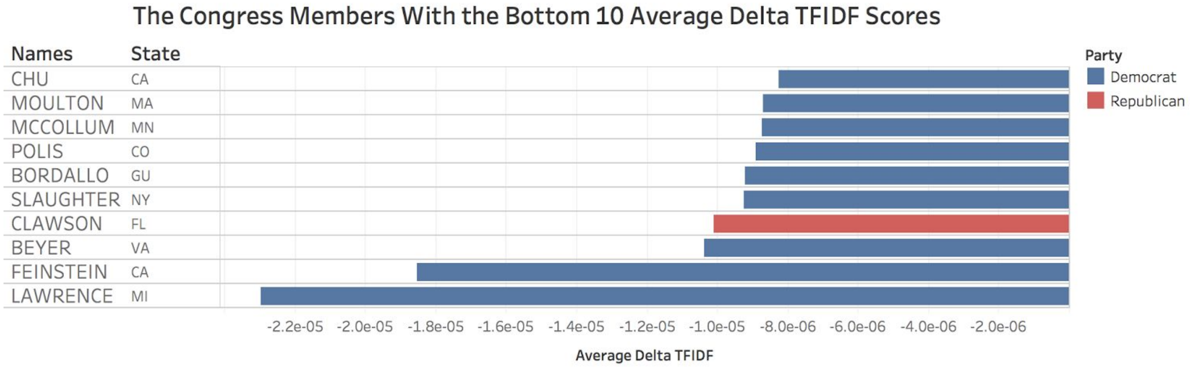


Fig. 6: Well known democrats are mostly at the bottom where we would expect

Nothing about this seems too unusual. Diane Feinstein is well known as being one of the most liberal members of the US senate, and Brenda Lawrence represents a strongly pro-labor district in detroit (an area with which there is likely to be little overlap with Republicans). To double check the thoroughness of this information though, we look at the DTF-IDF scores over time for these individuals.

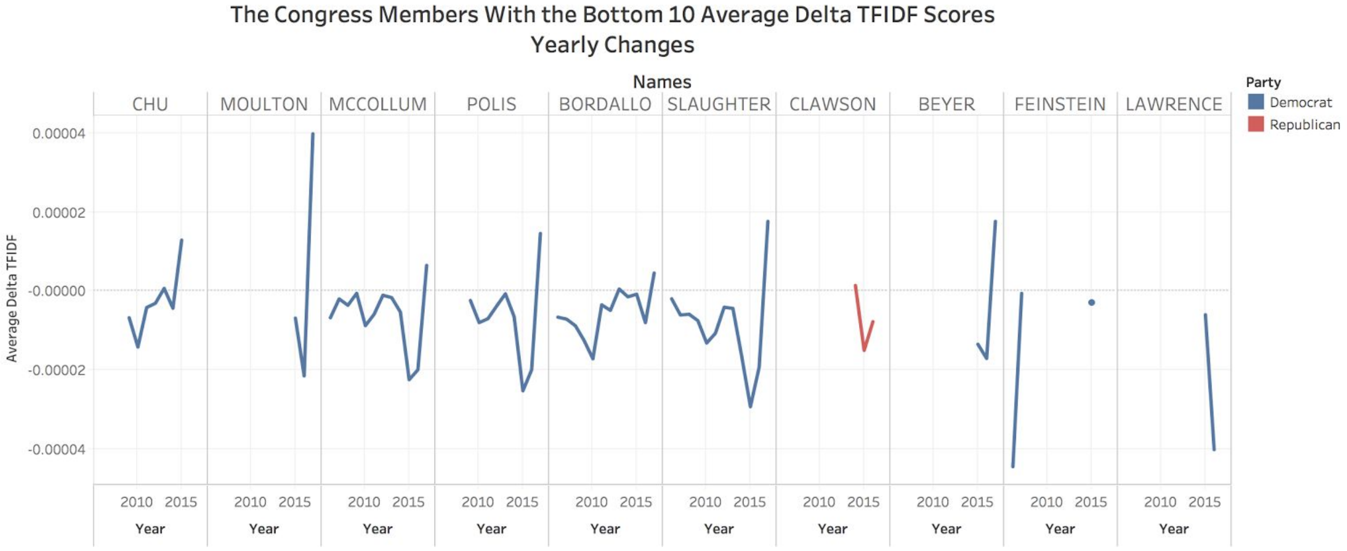


Fig. 7: Individuals at the top have the least amount of data available

We can see from fig. 7 that there is very limited information for the top 4 candidates, which likely explains their high positions more than anything else, particularly for Clawson, the lone Republican. Moreover the Republican version of this chart shows much the same thing. These results seem obvious when we look check the DTF-IDF distributions in fig. 8, however.

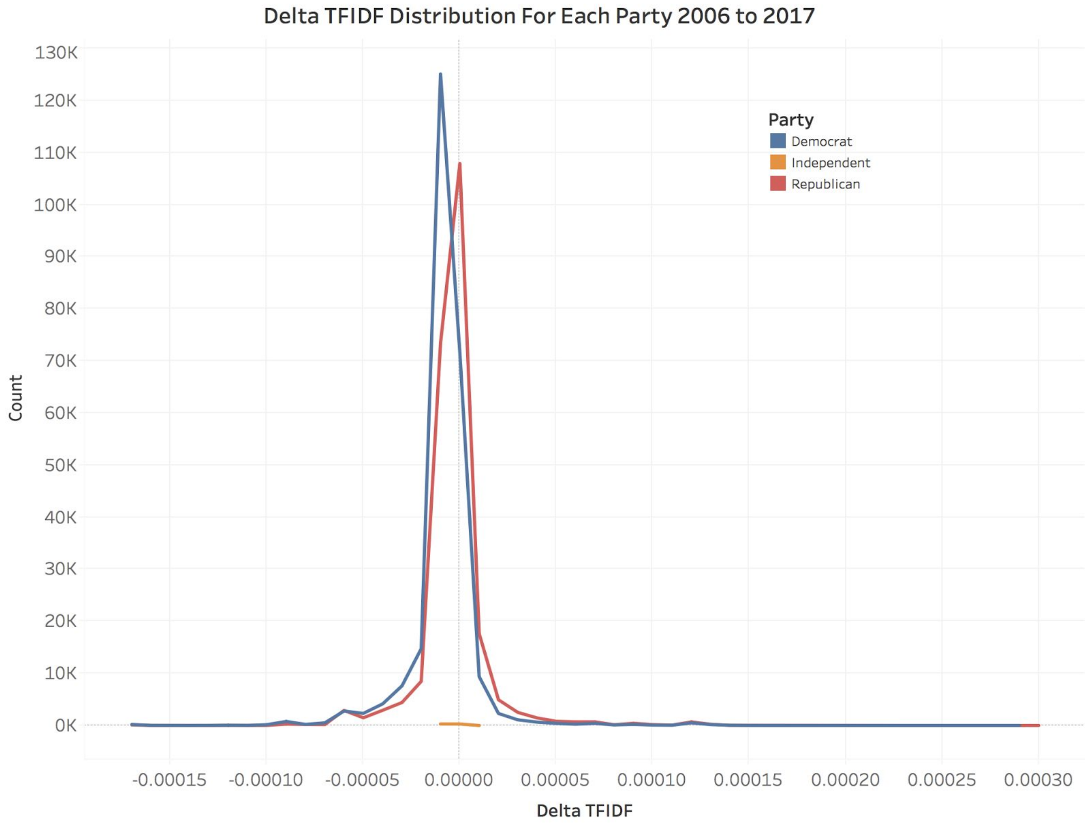


Fig. 8: There are very few people at the edges of the bell curve

There is a HUGE amount of overlap between the parties in terms of words used, as seen in fig.9. This confirms what we already knew (there there was a lot of noise in our data), but also makes examining the outliers rather suspect. Either because there was very little data available so we have too much variance, or because there is a lot of error. This overlap is doubly confirmed when checked with the most common words overall, as there is barely any useful information even after we removed the most common stopwords.

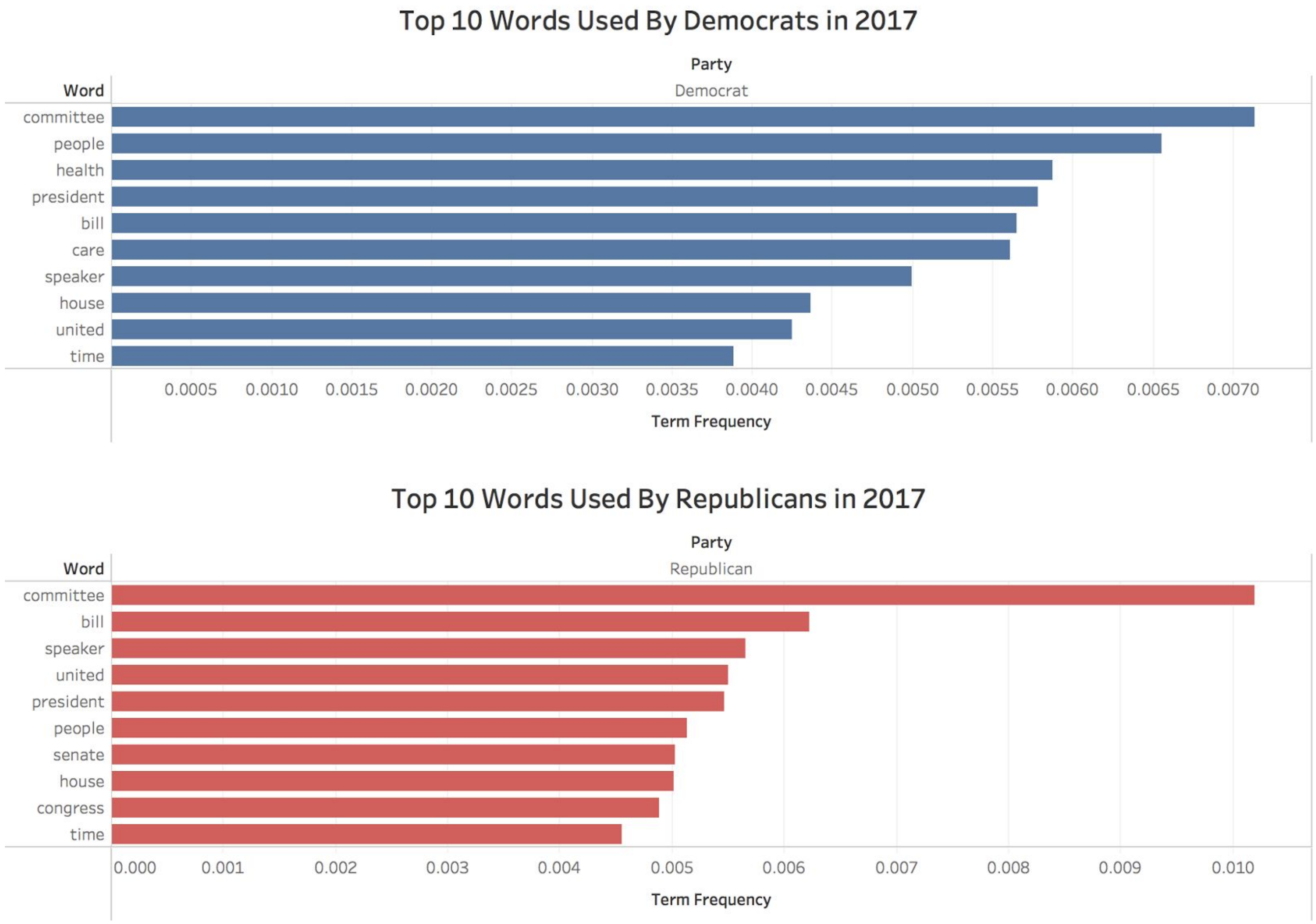


Fig. 9: There is significant overlap between the parties

This process has been fairly disheartening so far, but some of our results did lead to useful interpretation. Utilizing popularity polling data, we tracked DTF-IDF scores with approval ratings by state and linked them with the appropriate representatives. Despite some outliers, approval ratings greater than 50% seem to cluster around zero, which means more moderate individuals are being favored. For instance; Gary Peters has the lowest approval rating, at 38% with a DTF-IDF score of -2.7730e^{-06}, only 2% lower than the most democratic score. Meanwhile Bernie Sanders has the highest approval rating with 80% with a DTF-IDF score thats nearly zero.

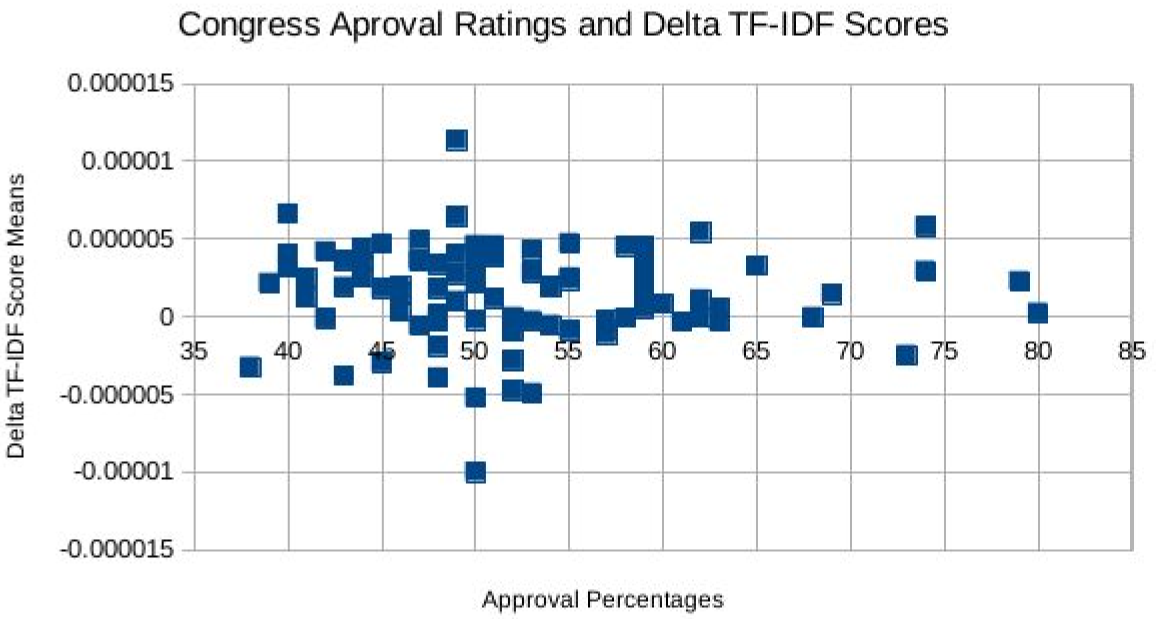


Fig. 10: DTF-IDF scores that are more centered correlate with higher approval

Conclusions

Unfortunately, we were unable to find significant variation between the language used by the two political parties. This is most likely the fault of our data cleaning which failed to remove all the non-partisan political speech and common buzzwords. That being said, while it is incomplete we were unable to find a similar publicly available data cleaning script, and so there is value in taking what we have done and building off of it. Moreover, while it is unwise to draw any hard conclusions given the nature of our data there seems to be a trend towards more moderate candidates having higher approval ratings, at least once they are in office.

Limitations

There are several severe limitations that should be covered regarding our preprocessing methods and analysis, that qualify our conclusions and the usefulness of our data. To begin with, we limited ourselves in the amount of data cleaning that we did after running into some early problems that appeared very difficult to solve. In particular, there was large amounts of noise generated from formal matters of congressional process (the beginning/end of a session, introduction of bills/individuals, recesses, etc.). This speech is both difficult to identify, and made worse by usually coming from the majority party. Additionally there are a lot of words that ran together in our initial parsing which we were unable to pull apart.

Additionally, Looking only at congressional floor records we miss out on a lot of the less formal activity that goes on in committee meetings and behind closed doors where much (if not all) of the actual lawmaking gets done. This means we cannot fully examine the motivation of individuals, given we are only capturing what they are willing to state publicly.

Regarding our analysis: *tfidf* by definition only looks at individual words, and thus lacks the detail and flexibility that n-grams or a more complicated model might have. We recommend that more work be done in all of these areas if future researchers are interested in this topic, with some specifics covered below.

Future Work

Our Goals

We are in the process of building the preprocessing scripts into a python library, since any analysis done using the congressional record will require parsing the data from PDF's, and it seems like a good public service to make our methods easily available. More thorough data cleaning is probably something that will go hand in hand with this, and is a higher priority than perfecting our analysis; more qualified people than us will possibly want to work with this data so we want to ensure it is as reliable as possible. This could involve more narrowly focused stopword removal, spelling correction, and better exception catching.

Recommended Further Work

There are a lot of superior models that could be implemented with this data, the most important facet of which is providing context. There are certain words (ex: "welfare") which mean wildly different things in different political contexts, and this needs to be addressed. This could possibly be done with sentiment analysis to determine which words/phrases are positive or negative within a particular partisan subgroup. Topics should also be taken into account, because words/phrases that are associated with the topic being discussed are not very predictive (ex: we do not care if people are talking about defense if everyone is talking about defense).

Ideally, we would also link our results up with voting patterns, proportion of attack ads in that election cycle, etc. To see if linguistic partisanship directly relates to behavioral partisanship.

Works Cited

Andris C, Lee D, Hamilton MJ, Martino M, Gunning CE, Selden JA. The Rise of Partisanship and Super-Cooperators in the U.S. House of Representatives, *PLoS ONE*, 2015. https://doi.org/10.1371/journal.pone.0123507.

Martineau, Justin and Finin, Tim. Delta TFIDF: An Improved Feature Space for Sentiment Analysis, *Proceedings of the Third AAAI Internatonal Conference on Weblogs and Social Media*, May 2009. http://ebiquity.umbc.edu/_file_directory_/papers/446.pdf.

Sources

1. Congressional biographical data: https://github.com/unitedstates/congress-legislators/tree/master/alternate_formats
2. Daily Congressional records: https://www.congress.gov/congressional-record
3. Popularity data: https://morningconsult.com/senate-approval-bernie-rubio-cruz/

Appendix A: Reflection

There are a lot of things we learned from this project (as one would hope), chief amongst them being 'do not spend three weeks collecting data without seeing if someone else has already done it.' That aside, the overall experience was one of constant pivots - with nearly everything we had planned on doing in the beginning needing to be scrapped. Our initial plan was to predict bias in news articles, which ended up with so many holes it is hard to cover them all. The primary flaw, however, was in how to label the articles for training purposes. In order to designate news sources as either liberal or conservative we needed a large source of text that was definitively one or the other, which proved to be very difficult to get even from sources which provided accessible API's. Moreover, within the news source, each reporter is different. A single reporter also was going to vary in political bias for each topic. So, we decided to find text we could give a more reliable label to: landing on congressional records due to how strongly left or right members of congress tend to be (and the large quantities of text that were publicly available for each person). Even so, we still planned on building some sort of predictive algorithm or usable 'tool'. This, unfortunately, also did not pan out as the data cleaning/extraction took nearly five weeks all on its own. Thus, we ended up with the purely exploratory project we have now.

We also wish we would have looked harder for previous data. After collecting our data, we found data from 2008 that was already cleaned. This would have been useful to have found earlier. Through finding this data, we found a lot of papers that used that data. They had either better methods, or errors that we could have avoided if we had found them earlier. We were so concerned with gathering data that we lost track of exploring it. Instead of focusing on a tool, we should have done an exploratory analysis from the get go. We spent too much time learning things that were not needed, but this would have been hard to stop due to how drastically our project changed.

Communication was also a big thing that we should have worked on. Dividing tasks among team members went smoothly, but there were a few times where people were unsure of what they were supposed to be working on. Most of this was due to our project change. We did a good job of using the strengths of every team member, which in the end, allowed us to efficiently construct solutions for many different issues in the project.

We could have better cleaned our data, primarily the parsing of PDFs. Although we could have scraped the web site, it was easier given our amount of time to download the pdfs and extract the text out of it. We also found out that filtering political text data using rudimentary stop words was not sufficient. Politicians use different vocabularies than individuals posting online reviews (which is what most stop word libraries target). We found many instances where we felt our solution could have been better had we cleaned these words out of our data. Analysis on whole sentences, as stated earlier in our paper, would have paved a way for a more sound solution. Parts of speech tagging would have allowed us to keep our stop words and analyze sentences, or statements easier.

Appendix B: Essential Code

B.1 Congressional Record Parser

```
def decodeText(text):
    """
    Input: A chunk of text.
    Output: The encoded version of that chunk.
    """
    try:
        return text.decode('unicode_escape').encode('ascii','ignore')
    except: # Special case where trouble encoding "\\" - file 2015-11-16
        text = "".join([t + '\\\\' for t in text.split('\\')])
        return text.decode('unicode_escape').encode('ascii','ignore')

def cr_processer(filePath1, filePath2, filePath3):

    """
    INPUT: three filepaths
    1. to a .txt file of the daily Congressional Record put out by the US Congress.
    2. path to legislators-current.json.txt - get from google drive
    3. path to legislators-historical.json.txt - get from google drive
    OUTPUT: dataframe of the parsed data, containing congress#, date, what was said
and the last names,
    party affiliation, and state of who said it. Note that all legislators with duplicate last
names
    have been remove for ease of analysis
    """

    #IF SYS.ARGV[1] DOESN'T WORK, USE 'FILEPATH' INSTEAD
    with open (filePath1, "r") as myfile: data = myfile.read().replace('\n', '')

    #FIND ALL THE NAMES
    names = re.findall('[M][s|r][\.][\s][A-Z]{2,}', data)
    names = [re.sub('[M][s|r][\.][\s]', '', x) for x in names]

    #FIND ASSOCIATED TEXT FOR EACH NAME
    split = re.split('[M][s|r][\.][\s][A-Z]{2,}', data)
    split = split[1:]

    #GET THE DATE OF THE RECORD
    date =
re.findall('(?:JANUARY|FEBRUARY|MARCH|APRIL|MAY|JUNE|JULY|AUGUST|SEPTE
MBER|OCTOBER|NOVEMBER|DECEMBER)[\s][0-9]{1,2}[\,][\s][0-9]{4}'
            ,data)[0]
    date = parser.parse(date)
    date = [date]*len(names)

    #GET WHICH CONGRESS IT IS
    congress = re.findall('[0-9]{3}', data)[0]
    congress = [congress]*len(names)
    congress = [x[0:3] for x in congress]

    record = pd.DataFrame({'text': split,'names': names, 'date': date, 'congress':
congress})

    #REMOVE UNICODE
    record['text'] = [decodeText(s) for s in record['text']]

    #REMOVE PUNCTUATION
    record['text'] = [r.translate(None, string.punctuation) for r in record['text']]

    #REMOVE NUMBERS
    record['text'] = [re.sub(r'\d+', '', x) for x in record['text']]

    #REMOVE NON-SPEECH BEURECRATIC CODES
    record['text'] = [re.sub(r'[A-Z]{2,}', '', x) for x in record['text']]

    #REMOVE EXCESS SPACES
    record['text'] = [re.sub(r'[[:space]]{2,}', ' ', x) for x in record['text']]
    record['text'] = [' '.join(word for word in x.split() if len(word)>3) for x  in record['text']]

    #REMOVE STRINGS THAT ARE OBVIOUSLY TOO SHORT
    remove = [len(x)<=30 for x in record['text']]
    record['remove'] = remove
    record = record.drop(record[record.remove==True].index)
    record = record.drop(['remove'], axis=1)

    #LOAD CURRENT LEGISLATOR INFORMATION
    with open (filePath2, "r") as myfile:
        current_legislators = json.load(myfile)

    #FIND PARTY
    party = [current_legislators[x]['terms'][len(current_legislators[x]['terms'])-1]['party'] for x
in range(len(current_legislators))]

    #FIND STATE
    state = [current_legislators[x]['terms'][len(current_legislators[x]['terms'])-1]['state'] for x
in range(len(current_legislators))]

    #FIND NAMES
    lnames = [current_legislators[x]['name']['last'].upper() for x in
range(len(current_legislators))]

    start_year = [current_legislators[x]['terms'][0]['start'][0:4] for x in
range(len(current_legislators))]
```

```
    leg_data = pd.DataFrame({'party': party,'state': state, 'names': lnames, 'start_year':
start_year})

    leg_data = leg_data.drop_duplicates(subset=['names'], keep='first')

    #LOAD PAST LEGISLATOR INFORMATION
    with open (filePath3, "r") as myfile:
        lh = json.load(myfile)

    lh=[x for x in lh if int(x['terms'][0]['end'][0:4])>=2006]

    party = []
    for i in range(len(lh)):
        try: party = party + [lh[i]['terms'][0]['party']]
        except:
            party = party + ['NA']

    names = [x['name']['last'].upper() for x in lh]

    state = [x['terms'][0]['state'] for x in lh]

    end_year = [x['terms'][len(x['terms'])-1]['end'][0:4] for x in lh]

    start_year = [x['terms'][0]['start'][0:4] for x in lh]

    hleg_data = pd.DataFrame({'party': party, 'names': names, 'state': state, 'end_year':
end_year, 'start_year': start_year})

    ld = leg_data.append(hleg_data)

    ld = ld.drop_duplicates(subset=['names'], keep='first')

    c_record = record.merge(ld, on='names', how='left')

    return(c_record)
```

B.2 TFIDF Calculator

```
returnTfCongressDataName <- function(df)
{
        if (is.character(df$text) == FALSE)
        {
                innerDf <- as.character(df)
        }
        else
        {
                innerDf <- df
```

```
    }

tidy_tempData <- innerDf %>% unnest_tokens(word, text)
data("stop_words")
cleaned_tempData <- tidy_tempData %>% anti_join(stop_words)
word_tempData <- cleaned_tempData %>% count(names,word, sort = TRUE)
totalword_tempData <- word_tempData %>% group_by(names) %>% summarize(total
= sum(n))
word_tempData <- left_join(word_tempData,totalword_tempData)
word_tempData <- word_tempData %>% bind_tf_idf(word,names,n)
word_tempData <- word_tempData %>% select(-total) %>% arrange(desc(tf_idf))
    return(word_tempData)


}
returnTfCongressDataDate <- function(df)
{
    if (is.character(df$text) == FALSE)
    {
         innerDf <- as.character(df)
    }
    else
    {
        innerDf <- df
    }

tidy_tempData <- innerDf %>% unnest_tokens(word, text)
data("stop_words")
cleaned_tempData <- tidy_tempData %>% anti_join(stop_words)
word_tempData <- cleaned_tempData %>% count(date,word, sort = TRUE)
totalword_tempData <- word_tempData %>% group_by(date) %>% summarize(total =
sum(n))
word_tempData <- left_join(word_tempData,totalword_tempData)
word_tempData <- word_tempData %>% bind_tf_idf(word,date,n)
word_tempData <- word_tempData %>% select(-total) %>% arrange(desc(tf_idf))
    return(word_tempData)


}
returnTfCongressDataCongress <- function(df)
{
    if (is.character(df$text) == FALSE)
    {
         innerDf <- as.character(df)
    }
    else
    {
        innerDf <- df
    }
```

```
tidy_tempData <- innerDf %>% unnest_tokens(word, text)
data("stop_words")
cleaned_tempData <- tidy_tempData %>% anti_join(stop_words)
word_tempData <- cleaned_tempData %>% count(congress,word, sort = TRUE)
totalword_tempData <- word_tempData %>% group_by(congress) %>%
summarize(total = sum(n))
word_tempData <- left_join(word_tempData,totalword_tempData)
word_tempData <- word_tempData %>% bind_tf_idf(word,congress,n)
word_tempData <- word_tempData %>% select(-total) %>% arrange(desc(tf_idf))
        return(word_tempData)


}
returnTfCongressDataParty <- function(df)
{
        if (is.character(df$text) == FALSE)
        {
                 innerDf <- as.character(df)
        }
        else
        {
                innerDf <- df
       }

tidy_tempData <- innerDf %>% unnest_tokens(word, text)
data("stop_words")
cleaned_tempData <- tidy_tempData %>% anti_join(stop_words)
word_tempData <- cleaned_tempData %>% count(party,word, sort = TRUE)
totalword_tempData <- word_tempData %>% group_by(party) %>% summarize(total =
sum(n))
word_tempData <- left_join(word_tempData,totalword_tempData)
word_tempData <- word_tempData %>% bind_tf_idf(word,party,n)
word_tempData <- word_tempData %>% select(-total) %>% arrange(desc(tf_idf))
        return(word_tempData)


}
returnTfCongressDataState <- function(df)
{
        if (is.character(df$text) == FALSE)
        {
                 innerDf <- as.character(df)
        }
        else
        {
                innerDf <- df
       }

tidy_tempData <- innerDf %>% unnest_tokens(word, text)
data("stop_words")
```

```
cleaned_tempData <- tidy_tempData %>% anti_join(stop_words)
word_tempData <- cleaned_tempData %>% count(state,word, sort = TRUE)
totalword_tempData <- word_tempData %>% group_by(state) %>% summarize(total =
sum(n))
word_tempData <- left_join(word_tempData,totalword_tempData)
word_tempData <- word_tempData %>% bind_tf_idf(word,state,n)
word_tempData <- word_tempData %>% select(-total) %>% arrange(desc(tf_idf))
	 return(word_tempData)


}
processDatedRecordsOfCongress <- function(df)
{
	mainData <- read.csv(df$file)
	mainData$text <- as.character(mainData$text)

	#name
	tData <- returnTfCongressDataName(mainData)
	tString <- paste(df$year,"TF_IDF_NAME.TXT")
	write.csv(tData,tString);

	#date
	tData <- returnTfCongressDataDate(mainData)
	tString <- paste(df$year,"TF_IDF_DATE.TXT")
	write.csv(tData,tString);

	#Congress
	tData <- returnTfCongressDataCongress(mainData)
	tString <- paste(df$year,"TF_IDF_CONGRESS.TXT")
	write.csv(tData,tString);


	#Party
	tData <- returnTfCongressDataParty(mainData)
	tString <- paste(df$year,"TF_IDF_PARTY.TXT")
	write.csv(tData,tString);


	#State
	tData <- returnTfCongressDataState(mainData)
	tString <- paste(df$year,"TF_IDF_STATE.TXT")
	write.csv(tData,tString);




	rm(mainData)
}
#processDatedRecordsOfCongress(tmp)
```

```
dateFile <- read.csv("dateAndFileList.csv")
dateFile$file <- as.character(dateFile$file)
dateFile$year <- as.character(dateFile$year)


for (val in 1:nrow(dateFile))
{


    processDatedRecordsOfCongress(dateFile[val,])
}
```

B.3 Delta-TFIDF Calculator

```
def tfidfDiffs(filepath1, filepath2):

    """
    Input: filepath1 = tfidf csv
             filepath1 = congressional record csv
    Output: congressional record with mean tfidf difference scores for each text
chunk
                added as a column
    """

    #LOAD IN TFIDF FILE AND GET WORDS THAT HAVE A NONZERO
SIGNIFICANCE
    dat = pd.read_csv(filepath1)
    dat = dat[dat['idf']>0]
    dat = dat[dat['party'].isin(['Republican', 'Democrat'])]

    #SPLIT WORDS BY PARTY
    repwords = dat[dat['party']=='Republican']
    demwords = dat[dat['party']=='Democrat']
    words = pd.merge(repwords, demwords, on='word', how='outer')

    #DROP UNNESSESARY COLUMNS
    tfidf = words.drop(['Unnamed: 0_x', 'party_x','n_x', 'tf_x', 'idf_x', 'Unnamed: 0_y',
'party_y','n_y', 'tf_y', 'idf_y'], axis=1)
    tfidf.columns = ['word', 'tfidf_r', 'tfidf_d']

    #FIND THE DELTA(TFIDF) BETWEEN REPUBLICAN AND DEMOCRAT FOR
EVERY WORD
    diff = [tfidf.loc[x]['tfidf_r'] - tfidf.loc[x]['tfidf_d'] for x in range(len(tfidf))]
    tfidf['diff'] = diff
    #LOAD THE CONGRESSIONAL FILE
    data = pd.read_csv(filepath2)

    #SPLIT EACH WORD CHUNK SO WE HAVE A LIST OF LISTS. MERGE THE
WORDS IN EACH WORD CHUNK
    #WITH ITS DELTA(TFIDF) SCORE
    text = [(re.split(' ', x.lower())) for x in data['text']]
    diffs = []
    for i in range(len(text)):
        df = pd.DataFrame(text[i])
        df.columns = ['word']
        y = tfidf.merge(df)['diff']
        diffs.append(np.mean(y))

    data['tfidfd'] = diffs

    return data

def main():
    # Call the function and have the script save a csv:
    tfidfDiffs(sys.argv[1], sys.argv[2]).to_csv('cong_record_tfidfDiff.csv')
    print "You can find your csv in the cwd folder with the name
cong_record_tfidfDiff.csv"

if __name__ == "__main__": main()
```

Appendix C: Additional Plots

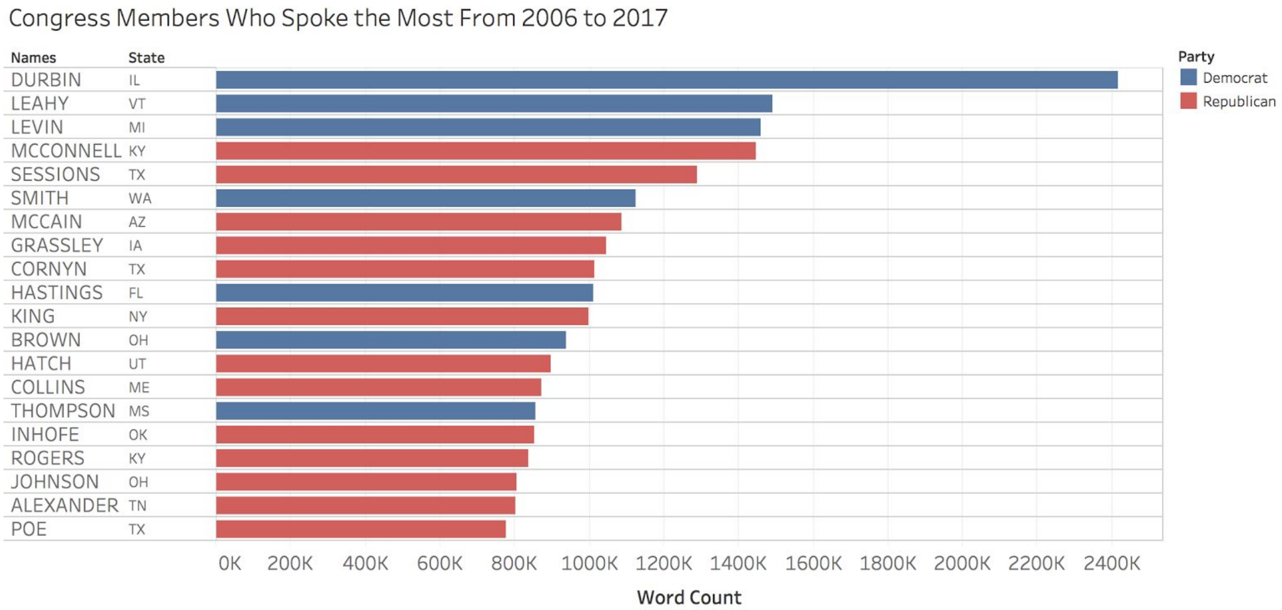


Fig. C.1: Richard Durbin of Illinois likes to talk...a lot

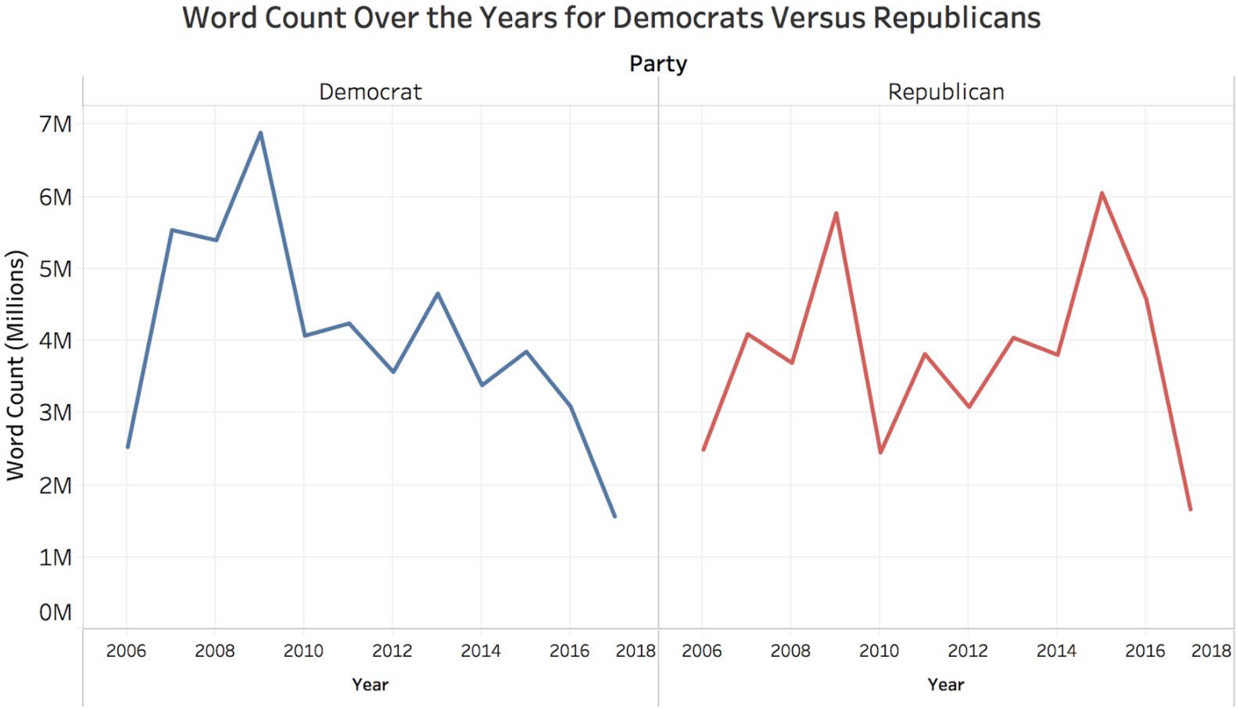


Fig C.2: Word counts mostly mirror each other, with 2015 as the exception (pre-election)