Predicting How People Vote From How They Tweet

A Mathematics Research Project submitted to
The Master of Arts in Teaching Program
of
Bard College

by
Rao Vinnakota

Annandale-on-Hudson, New York
May, 2019

Abstract

The project observes tweets concerning the midterm elections and uses the sentiment drawn from those tweets to predict voting trends.

Contents

Dedication

Text of dedication.

Acknowledgments

Text of acknowledgments.

1
Introduction

Donald Trump stunned the world when he was elected President of the United States. Trump ran an unorthodox campaign, specifically in regards to his use of social media. Trump used accounts on Twitter and Facebook as a voice to attack candidates, disseminate campaign information, and engage with voters.

Public polling in 2016 didn't predict the upset. In fact, it didn't come particularly close. The latest tracking polls before election day showed Clinton in the lead with a margin of anywhere from +1 to +6. Most pollsters believed that Trump would sweep strictly conservative areas, but Clinton would every state that really mattered. Obviously, that didn't happen

If you look through tweets following the election, Trump supporters claimed that victory was inevitable, and that Trump's social media interaction demonstrated it. true? Was Trump's potential victory present on social media for people to see?

That is idea my senior project explore. Can you mathematically evaluate opinions on social media to predict voting trends. I will gather tweets pertaining to the 2018 midterm elections and use the sentiment expressed in those tweets and attempt to predict voting results.

1.1 Political Polling - A Short Overview

The earliest iteration of polling was a straw poll conducted in Harrisburg, Pennsylvania which showed Andrew Jackson leading John Quincy Adams in the 1824 Presidential Election. Journalists from The *Harrisburg Pennsylvanian* asked local voters which they preferred. Jackson winning both the polls and the election created legitimized the poll. Polls continued to be local until the early 20th century when Literary Digest sent out postcards to reader asking for their pick of the presidential election. The Literary Digest correctly predicted 5 straight elections and opinion polls reached the national stage.

In the advertising boom of post-World War II, most polling shifted to telecommunications. With telephones, there was an ability to reach a wider audience, ask deeper questions. Pollsters could now build a profile of a voter, and forecast more accurate results. Accompanying the advance in polling was a focus on sampling. Pollsters couldn't call every single house in the United States. Instead, groups of people or "samples" that would accurately represent the country were called. Responses from the sample were extrapolated to cover the whole.

Today, the digital age and social media have made polling's barrier of entry low. A short scroll through Twitter will show hundreds of "opinion polls". Youtube videos' likes and dislikes function as an informal poll. The Algorithm will often promote videos that are more "well-liked". But, professional political polling is still done along the same lines. A population is sampled, called, and profiled. Profiling technique has improved. Today, forecasters like 538's Nate Silver, will build complex profiles of voters in various regions. But, the source of the information and the method of obtaining the information remains the same.

1.2 Social Media, Twitter, and Politics

The internet's first foray into politics was in 2008. Barack Obama established that it was acceptable for a candidate to leverage the internet as a resource, using the internet as a tool for both message and money. The website MyBarackObama.com helped Obama set records in grassroot donations and mobilizations.

Social websites from the forums of the early 2000s to social media platforms that are popular today are the most visited websites on the internet. These forums encourage conversation, including the discussion of politics. Especially for younger voters who don't own any landlines, these forums are the few places they express political opinions.

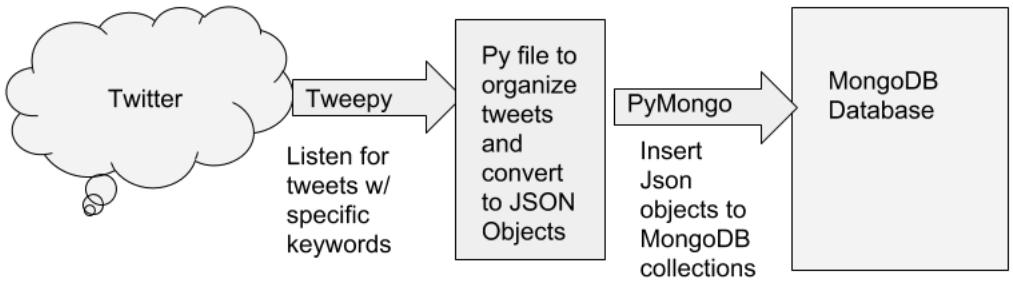
This project will specifically look at political opinions expressed on the microblogging platform Twitter. What sets it apart from blogs or other political forums is their ease of use. The longform opinions that are on forums and blogs are short and un-edited in Tweets. Twitter's mandatory message format - less than two hundred and eighty characters - makes sure posts are similar in style, and are far easier to analyze.

1.3 Big Data

A decade ago, attempting to gather, clean, and analyze millions of tweets would have been a fool's errand. It's with big data techniques and packages that this project is now possible. This presents my experiment with several advantages. Unlike traditional pollsters, I won't need to design a sample, but simply collect every tweet I can find. I also will capture a different audience.

2
Building a Data Pipeline

Just under three million tweets were collected for this experiment. These tweets were streamed from Twitter through a collection pipeline and stored in a Database in the weeks leading up to the Midterm Election. The pipeline was organized as such:



Building the pipeline consisted of three distinct tasks.

1. Building a tweet streamer to "listen" to tweets based on keywords
2. Preprocessing the tweets
3. Storing processed tweets in a database.

All the code written to built the pipeline can be found in Appendix A.

2.1 Streaming Tweets

There are two ways to gather tweets using the Python Twitter API (Applications Package Interface). One is to query a profile or a search term. This method is not preferred for several reasons. First, Twitter has put a hard cap on the number of tweets each query can return at one time. This means that querying will be in-exact and time consuming loop. Next, Twitter's search function will naturally promote tweets that have more engagement. This means that tweets with more likes and retweets, comments, and followers will get preference, which results in a biased data set.

The other way to gather tweets is to stream tweets. A Tweepy stream monitors live twitter content and returns all tweets that contain a given set of keywords. Streaming has the disadvantage of gathering only live tweets - it does not search tweets from before it started running. But, streaming has the advantage of gathering any tweets that meet the keyword requirement, irrelevant of engagement. Consequently, it was decided that the tweets gathered would be through streaming.

The streamer is abstracted in the Twitter API. To use it, the user must define the actual methods the streamer needs to find and save tweets. The methods defined here are on_connect, on_error, and on_data. The first two are trivial. On_connect is used to connect, and confirms once the stream is live. If the stream crashes - which can happen for several reasons - on_error informs the user why the stream stopped (memory, exceeding latency, etc.). The last method on_data is where the tweets are received and processed. Each tweet is received as a dictionary

data structure where keys are attributes of the tweet (i.e. date and time, text, user, etc.). From there any processing is up to the user; the tweet's text can be printed, the tweets can be prepared for entry to a data base, etc. It's here that the tweets will be modified to fit MongoDB's insertion requirements.

2.2 Databases

The tweets that the Tweepy streams need to be stored. An Excel file is a convenient but imperfect solution. A database will take full advantage of the object stored as well as the large data set in question.

2.2.1 Overview

The term database refers to any structure that electronically stores data. In extension, the Database Management System is the program that allows users to interact with the database. In this paper, when the term database is used, it refers to both the structure and the system.

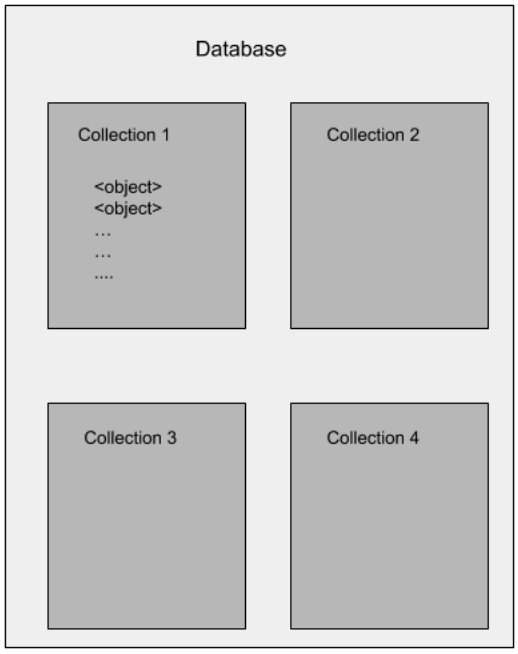
There are several types of databases. These types are classified by the way the data is organized inside the structure. The relational database model uses tables with rows and columns to store information. Relational databases use SQL to organize, extract, and insert data. Non-relational databases, referred to colloquially to as "NoSQL" use an object-oriented approach with different query languages.

Databases offer several advantages that other storage options like Excel or CSVs will not. The ability to quickly search and sort through millions of tweets as well as the flexibility to manipulate said data is something that isn't available to Excel. Additionally these databases have organized and maintained APIs for Python.

2.2.2 MongoDB

MongoDB is a popular non-relational database management system thats's offered for free. It has an efficient query language as well as a well-defined API Pymongo. In a MongoDB database,

each tweet is an object. These objects are stored in collections and the collections are stored in a database:



These JSON objects have fields, populated with values. MongoDB allows a user to query collections using field values. These queries can be exact matches or other operators. For example, you can query tweets by date, or location (by exact match). You can also query sub-matches. That means conditions within a field and not an exact match, like searching for all tweets that contain Donald Trump.

MongoDB has a mongoshell through which a user can interact with the databases. For longer and larger actions, we use the Python API Pymongo to write python scripts. Pymongo allows us to insert, edit, and query databases and manipulate the output.

For more specifics on Pymongo and setting up MongoDB, refer to the Appendix or the Pymongo documentation.

2.3 Data Collection

Implementing the pipeline referred to earlier means that tweets are streamed and inserted into a MongoDB Database. The first step of streaming requires search terms. These search terms were taken from house and senate races, as well as general terms that refer to the 2018 midterms. Search terms for Senate and House Races were tweets about the candidates as well as a hashtag referring to the race.

To direct the tweets streamed into a database, the on_data method is defined. This method is designed for the streamer to deal with the tweets it selects.

```
def on_data(self, data):
try:
            client = MongoClient(MONGO_HOST)

            # Use twitterdb database. If it doesn't exist, it will be created.
            db = client.miscdb

            # Decode the JSON from Twitter
            datajson = json.loads(data)

            #grab the 'created_at' data from the Tweet to use for display
            created_at = datajson['created_at']

            #print out a message to the screen that we have collected a tweet
            print("Tweet collected at " + str(created_at) + " " + datajson['text'])

            #insert the data into the mongoDB into a collection called twitter_search
            #if twitter_search doesn't exist, it will be created.
```

```
            db.misc.insert(datajson)
        except Exception as e:
           print(e)
```

The tweet is converted into a JSON object - MongoDB's preferred type - and inserted into a database and collection of the users choice. This is a barebones definition. Sorting, editing, and adding information to tweets would all be defined here. For the entire script that streamed tweets for the senate or the house, refer to Appendix A.

The data collection process ran through three separate scripts each with a different set of search terms. Those three scripts fed into three different collections in a massive database. Together they assemble a diverse dataset that contains just under three million tweets. Chapter four will explore the dataset we've assembled.

3
Sentiment Analysis

Sentiment analysis is the computational approach to evaluating text, which includes determining both sentiment (positive/negative) and intensity. There has always been an interest in the idea of sentiment analysis or objectively evaluating text for polarity. But, the field has seen a burst of activity in recent years due to a few key factors - the availability of text through the world wide web, the advance of big data, and the improvement in machine learning techniques.

The question of sentiment analysis's real effectiveness is often brought up. The general consensus is that a ceiling does exist for sentiment analysis, much like a ceiling exists for human analysis. There are statements where humans disagree on the sentiment. The ceiling is statements that have near unanimous agreements. If most humans can agree on a statements, it's not a stretch to assume that an algorithm could come to a similar conclusion.

This paper aims to use sentiment analysis to "poll" twitter users. The model built will make the trade-off specific targeted questions for a large sample size. To make use of collected data, it's important that the sentiment analysis algorithm used is effective and fast; after all, it has to evaluate three million tweets.

3.1 Lexicons

The basic idea of sentiment analysis is really simple. The algorithm recognizes positive and negative words, and then finds a method to average these words and determine an overall sentiment and intensity for the statement. But how does it know which words are positive and which words are negative? It uses a lexicon.

Lexicons are dictionaries. Large dictionaries which contain a list of positive and negative words, each with an intensity expressed. They allow algorithms to quickly ascertain the meaning of a word without having to re-define the word each time. Several well-received lexicons include the LIWC, Bing Lui, and the Harvard Inquirer. The lexicon that we'll use is known as VADER, which will be thoroughly examined later in the chapter.

Lexicons can be broken into two types - semantic orientation or polarity based, and sentiment intensity or valence based. These general classifications cast a wide net and almost all lexicons fit in one or the other.

A well-known polarity-based lexicon is the Linguistic Inquiry and Word Count or LIWC (pronounced "Luke"). LIWC organized words into seventy six categories and has about 900 words that are organized into two categories that are associated with emotion. It's the results of efforts by experts in psychology, sociology and linguistics. It's seen as arguably the best lexicon available, but does not recognize acronyms, abbreviations, or slang. These are all large parts of analyzing social text, and LIWC falls short.

The original valence-based lexicon is The Affective Norms for English Words or ANEW. It created ratings for over one thousand words of the english dictionary on a scale from one to nine, with a neutral midpoint at five. Words with a value greater than five will have a positive and pleasant meaning and negative and unpleasant otherwise.

3.2 VADER

The algorithm used to evaluate the data set is known as the Valence Aware Dictionary for sEntiment Analysis or VADER. It was developed by a pair of researchers CJ Hutto and Eric

Gilbert at the Georgia Institute of Technology for the express use of evaluating valence in social media.

Social media and microblogging in particular poses a challenge to to sentiment analysis algorithms. This is the result of several problems. Earlier lexicons and algorithms were developed before social media's usage rates grew exponentially. They simply cannot handle the rate and volume that social media content is produced. Next, microblogging content is short. Twitter is limited to just two hundred and eighty characters. The average tweet is no more than a sentence or two. This limits context clues for traditional algorithms to pick up. Lastly, social media is famous for its abbreviations. Phrases like "LOL", "IRL", and "TBH" are commonplace on Twitter, but not commonplace on lexicons developed for traditional text. These are problems that VADER solves.

3.2.1 Construction

Hutto and Gilbert had four goals when they constructed the lexicon and the algorithm:

1. The algorithm performs effectively on social media text, but can still be generalized to all media sources.

2. The algorithm should have no training required from the user, but should still be based on a human created and cured lexicon.

3. The algorithm should be fast enough to use online.

4. There's no need to make a speed-performance tradeoff.

A basic overview of the VADER construction process is as follows. Hutto and Gilbert took the LIWC lexicon and made every single term in the lexicon a candidate for a new lexicon. Then, they proceeded to add social media specific terms which include abbreviations, phrases, and emojis as new candidates. These terms were trimmed and weighted in several stages until the final lexicon was produced.

The first stage of trimming used Wisdom-of-the-Crowd. WotC uses a large number of people to evaluate words in the lexicon. The large numbers combats the inexperience of the human raters themselves. Terms with a non-zero valence score were eliminated. Terms with a standard deviation greater than 2.5 which indicates that crowd couldn't reach a reasonable consensus were also eliminated. The terms were left had both a sentiment and an intensity. Terms such as "good" and "okay" while both positive, had a difference in score distinguishing their different meanings.

Their next stage used professional raters to identify parts of texts that can modify intensity despite not necessarily having an intensity themselves. These heuristics includes punctuation, verbal modifiers, and capitalization. Other heuristics identified were terms like "but" that signals a shift in sentiment and negated sentiment.

The last stage used professional raters to distinguish specific intensity modification of heuristics. This was done by taking a tweet, modifying either the syntax or the punctuation, and then randomly inserting these modifications into the dataset for the raters to evaluate. That way the algorithm could put a score on the change in intensity.

3.2.2 Effectiveness

The algorithm's effectiveness can only be evaluated by comparing it to the ground truth. The research team obtained ground truth statements for four domains - social media, movie reviews, product reviews, and editorials.

VADER's lexicon's competence was compared to several other well known lexicons such as LIWC, Bing Liu, Harvard Inquirer's, and more. This comparison made sure to use the same rule-based model for all of the lexicons. The three factors to evaluate the lexicons were:

- The correlation of computed intensity to the ground truth intensity.
- Precision - the number of items correctly classified divided by the number of total items.
- Recall - the number of items correctly classified divided by the number of items in the particular class

- F1 Score - the harmonic mean of the precision and the recall.

VADER outperformed some of the best lexicons in the industry currently. The results show that VADER's correlation was consistently higher, and it had the best F1 score for tweets. Hutto and Gilbert concluded that the reason was incorporating human heuristics into their model, essentially balancing a quantitative and qualitative approach.

3.2.3 Using VADER

VADER was written in Python, and is available to be implemented in Python. Below is how to use VADER to evaluate a single tweet.

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer


analyzer = SentimentIntensityAnalyzer()
analyzer.polarity_scores("I'm so happy!")
```

This piece of code will return the following output:

```
{'neg': 0.0, 'neu': 0.318, 'pos': 0.682, 'compound': 0.648}
```

The package returns a series of scores, breaking down how the various components of the model's final score. Positive tweets are tweets that score greater than zero and negative tweets score less than. The intensity is determined the absolute value of the score.

A few interesting observations to note:

- Sarcasm is impossible for the algorithm to understand. To be fair, many humans also struggle with recognizing sarcasm in text.

- Any tweet that's an observation tends to be neutral, even if that observation has a positive or negative slant.

3.3 Building a Sentiment Profile

Each candidate in a given race has tweets about them. Using VADER, we can build a sentiment profile for them. The sentiment profile contains five statistics:

- Total Number of tweets
- Number of Neutral Tweets (Compound = 0)
- Number of Positive Tweets (Compound > 0)
- Number of Negative Tweets (Compound < 0)
- Average Compound Score

The sentiment profile will be the base of the features we'll use to model. Below is a snippet of code that calculates the profile for a given user.

```
sentiment = analyzer.polarity_scores(tweet)
if (sentiment['compound'] == 0):
neu_tweets += 1
if (sentiment['compound'] > 0):
pos_tweets += 1
if (sentiment['compound'] < 0):
        neg_tweets += 1
ave_score += sentiment['compound']
count += 1
```

This snippet exists in a for loop that iterates through all the tweets found about a candidate. For the full script, refer to the Appendix.

Here's an example of a sentiment profile:

Candidate	Race	Count	Pos Tweets	Neg Tweets	Neu Tweets	Ave Compound
Ted Cruz	Texas Senate	111459	39642	33701	38116	0.0147602033
Beto O'Rourke	Texas Senate	34926	11445	3830	19651	0.1143955134

These profiles exist for every single candidate that participated in the election. You can find the sentiment profiles for house and senate candidates in Appendix C. The profiles are the data that will feed into the models that we'll build.

4
Exploring the Dataset

At the end of Chapter 3 we discussed building sentiment profiles for a candidate. These profiles are inspired by the profiles that many polling places create as they ask questions on their candidate. To run with the inspiration, this chapter dives into the dataset that we've created and examines if trends found in polling are reflected on twitter.

4.1 Races Collected

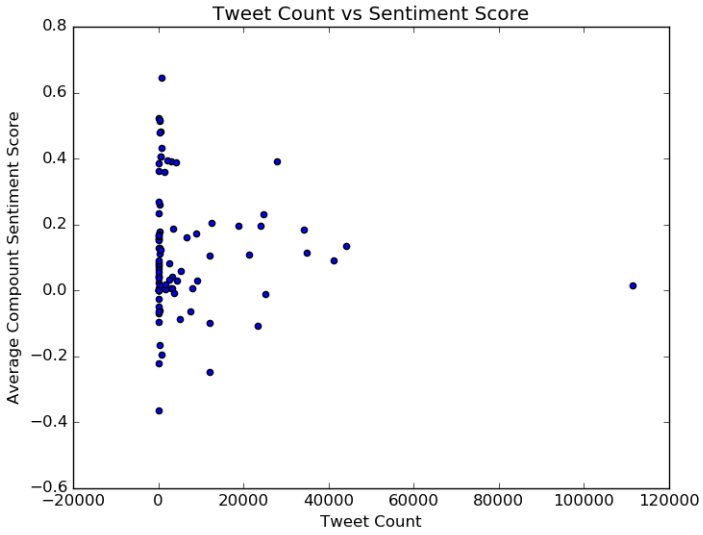
In the data gathering process, tweets were collected from thirty-three senate races, twenty nine house races, and a series of miscellaneous hashtags designed to capture national sentiment about the election.

After counting tweets and more, displayed below are leaderboards for house and senate. The most popular races consisted of the races with the most tweets and engagement. The Most Tweets category refers to the candidates that were tweeted about the most. The Best Liked goes to the candidate that had the highest portion of positive tweets. Least liked and Most Boring work the same way, except for Negative and Neutral tweets respectively. There was a five hundred tweet minimum to qualify for the leaderboard.

Rank	Most Popular Race	Most Tweets	Best Liked	Least Liked	Most Boring
1	Texas Senate	Ted Cruz	Angela Green	Josh Hawley	Tony Campbell
2	Florida Senate	Rick Scott	Lawrence Zupan	Angus King	Kevin Cramer
3	Indiana Senate	John James	John Barasso	Bill Nelson	Matt Rosendale

Rank	Most Popular Race	Most Tweets	Best Liked	Least Liked	Most Boring
1	NY-22	Troy Balderson	Xochitl Small	Josh Hawley	Lizzie Fletcher
2	VA-07	Steve Knight	Harley Rouda	Angus King	John Culberson
3	KY-06	Mike Bishop	Ross Spano	Bill Nelson	Gil Cisneros

One interesting point is that those who lead the various sentiment-based categories were no where close to the leaderboard in the most tweets category. Generally it felt that if a candidate was tweeted about more, their average sentiment score tended to drift towards neutral with the positive and negative tweets balancing out. Here is that relationship in graph form.



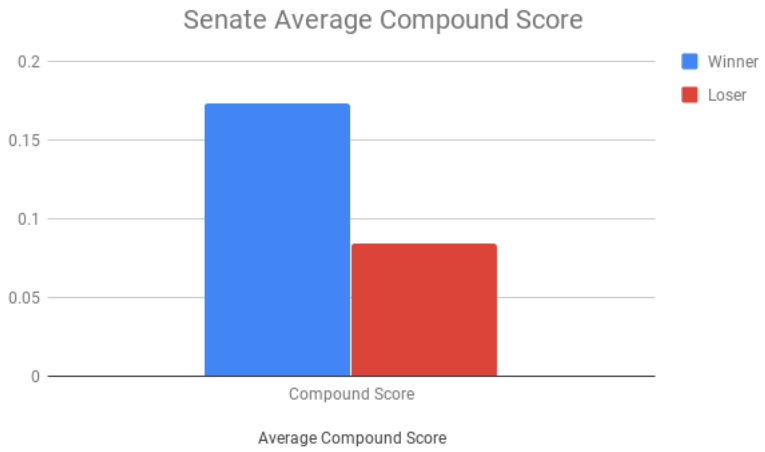
As the count increases, the range in sentiment scores decrease. The samples with tweet counts greater than twenty thousand all stay within a range of 0.4 on either side of neutral. Conversely, the range for sample sizes less than twenty thousand is from -0.4 all the way to 0.7. Essentially, the larger sample sizes are more balanced with a number of positive and negative tweets. That lends support to the idea that our sample isn't overly biased in one ideological direction.

4.2 Comparing Sentiment

4.2.1 Winners vs Losers

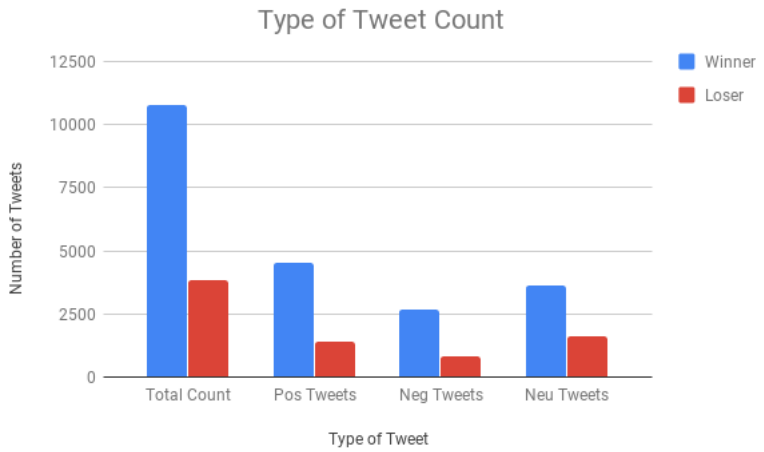
The point of an election is to win. The question everyone wants to answer is how? What's separates the winners from the losers, and how is that reflected on Twitter? Let's take a look at all the features from the sentiment profiles we created in Chapter 3. We'll begin with the senate.

The graph below shows the average compound sentiment score for winning and losing Senate candidates.

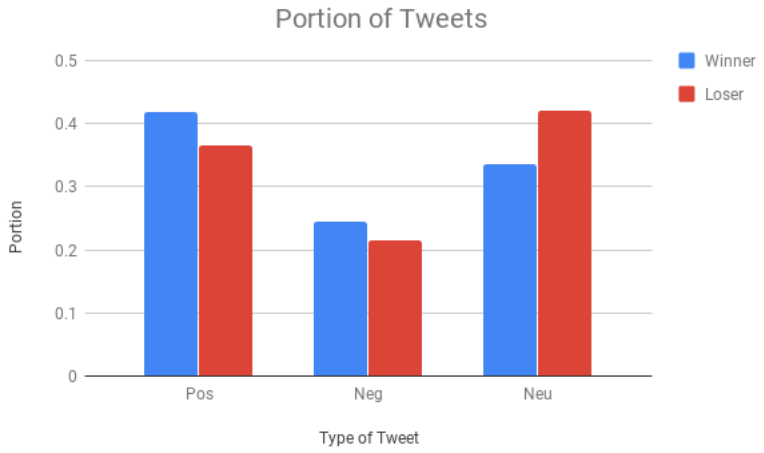


The winning candidates are heads and shoulders above the losing candidate, and have a much better sentiment score as a whole. That feels right. Candidates that win elections are favored by the voters and that should be reflected on twitter.

Compound score is a combination of positive and negative tweets. If winners have a higher compound score, it stands to reason that losers will have more negative tweets than winners. The next figure breaks down the count by type of tweet - positive/negative/neutral.

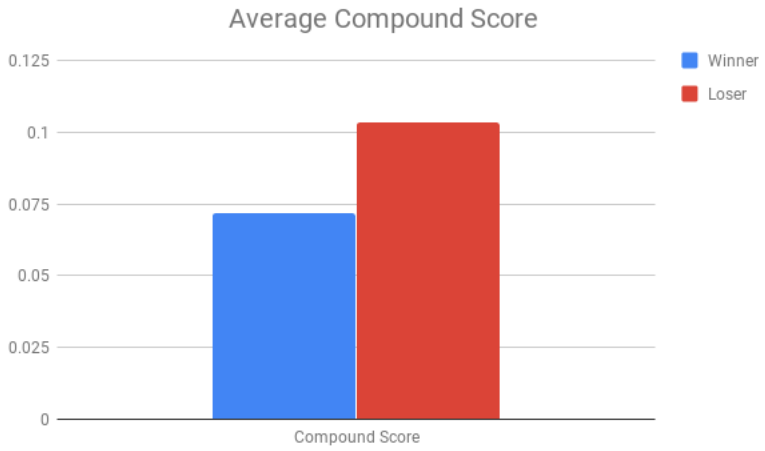


The election losers don't actually have fewer negative tweets. That's probably because there's just so many more tweets regarding winners than there are losers. To get a better view, let's break down types of tweets into percentages of the total count. That is to say, what percentage of a winner's tweets are positive, negative, or neutral?

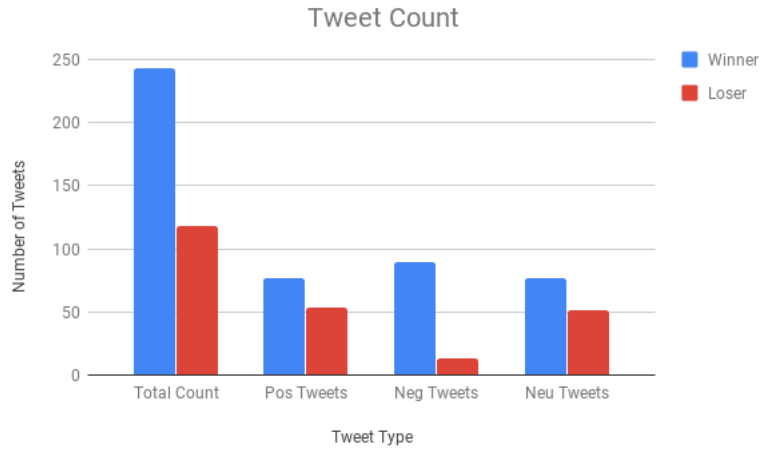


A closer look shows that the real difference lies in the fact that losers have a greater portion of neutral tweets than winners do. In therms of positive and negative tweets, the portions differ by less than five percentage points. Neutral tweets have a sentiment score of zero which would dilute and lower the average compound scores. Winners are not only tweeted about more than losers, but they're more likely to have tweets that show *some* sentiment. But do these patterns hold true for the house?

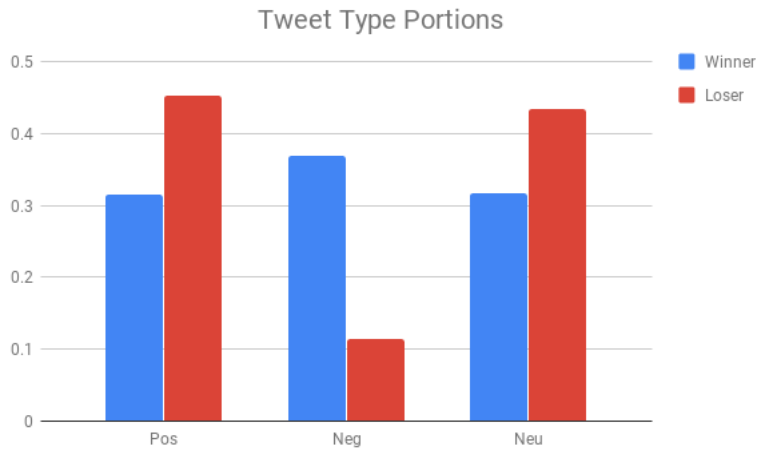
The average compound scores for winners and losers are displayed below:



Unlike the senate, the losers had a higher average sentiment score than the winners. That's unexpected. Let's take a deeper look into how those numbers break down.



Winners do have more positive tweets, but the large difference in the number of negative tweets is what ultimately influences the lower average compound score seen. Unlike the senate, the house races show an even-ness when it comes to neutral tweets. These numbers are hardly conclusive, partially because the large difference in tweet count is inflating these numbers. Here's a breakdown of the tweet portions:



The house and the senate winners both roughly have 30% of their tweets designated neutral. The real difference is that negative tweets occupy a far higher proportion of tweets. It doesn't feel that positive and negative tweets have a distinguishable effect on winning. In fact, the only trend that holds across both house and senate winners is the fact that winners are just tweeted about more. The house had a ratio of about 2-to-1 while the senate had 3-to-1. Bottom line, positive tweets are preferred, but overall engagement is key.

4.2.2 Democrats vs Republicans

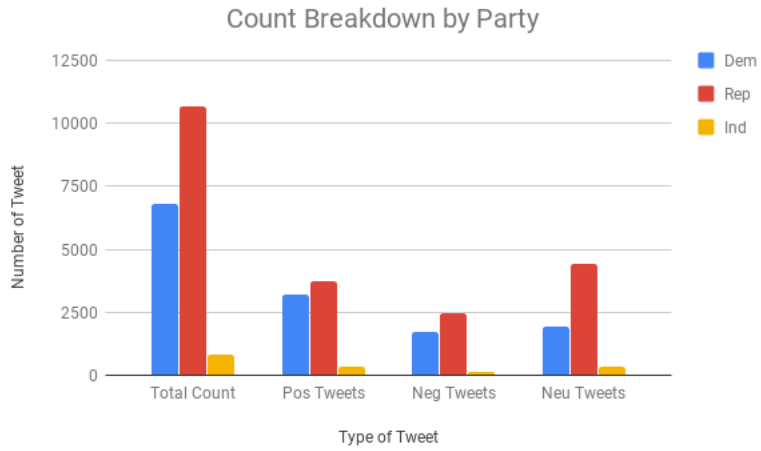
The two party system is a large part of our election process. The parties command massive influence and many voters choose a candidate based entirely on party affiliation. This means that a party's national platform and performance can influence their performance in more local elections like the house and the senate.

In the house, the Democrats flipped a number of Republican Incumbents taking a majority. In the Senate, the Republicans held the majority thanks to the uneven split in seats up for election.

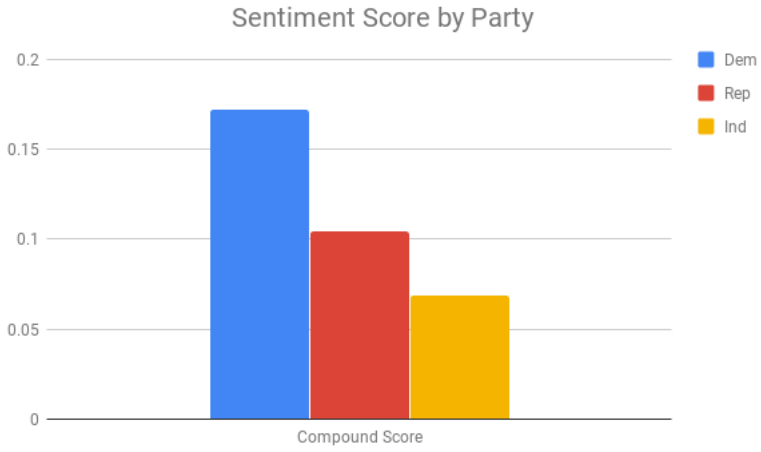
Incumbents	House	Senate
Democrats	1	24
Republicans	28	9

Winners	House	Senate
Democrats		
Republicans		

In the previous section, it showed that overall sentiment and tweet counts were two features that winners of elections had. If those traits hold, it should show that Democrats in the house should have a higher average sentiment and more tweets and the same for Republicans in the Senate.



The Republican dominated Senate did indeed carry the higher total count. They also led in every single other category. This will have interesting effects on the actual sentiment comparison.



Republicans leading both in positive and negative tweets meant that they suffered in the overall sentiment tally. The winning party held the tweet count lead, but not overall sentiment. If tweet count is truly the major feature to follow, the trend should hold in the house

4.3 National Trends

This data set presents the opportunity to explore how national trends are reflected on twitter. Not only that, it'll give a better idea on the kind of sample collected and how it reflects to the samples pollsters take when they observe overall trends they expect to influence elections.

4.3.1 Large National Trends

4.4 A Case Study: The Texas Senate

Appendix A
The How-To Guide

The following chapter is an instruction guide to recreate experiment in all parts from gathering tweets to creating simple models in Python. If you're looking for full scripts, those can be found in Appendix B. A quick disclaimer. All the instructions that follow are assuming that you are working on a UNIX system such as MacOS or Ubuntu.

A.1 Installation

There are several programs and packages that need to be installed to gather tweets as well as running a model:

- Python3 - If you're working on a UNIX system, this should come pre-installed. If it doesn't, install it using your system's package installation manager.
- MongoDB - A database management system. Find the distribution for your operating system on the website.
- Text Editor - Choose one that best meets your needs.

There are several python libraries that need to be installed for the scripts in Appendix B to work. They're listed below:

- Tweepy - The Twitter Python API
- Pymongo - The MongoDB Python API
- vaderSentiment - The python library that contains the VADER algorithm discussed in Chapter 3
- Sci-Kit - A Machine Learning Library used to build machine learning. models
- ConfigParser - A python package used to read and parse through configuration files

Python libraries are installed using pip. From the command line, execute the command:

```
pip3 install <library>
```

A.2 Gathering Tweets

A.3 Building Datasets

A.4 Building Models

Appendix B
Code

This project required a number of python scripts to be written. These scripts covered collecting data, organizing and cleaning collected data, as well as building predictive models. Below, you'll find every script written along with its intended purpose.

***

Name: Listener

Purpose: This script is a filled in template of the code written to stream tweets into a database. The "misc" in the title refers to the fact that this version of the script is streaming tweets from the list of miscellaneous search terms. It has two very similar scripts for the senate and the house.

Execution: It requires a server with the mongod daemon running as well as a Twitter Application with an API key/secret pair. Python packages used are Tweepy and Pymongo. Can be run directly from the command line or through an IDE.

Expected Output: Will print tweets as they are streamed. Tweets themselves will be stored in the targeted database.

```
1 from tweepy import Stream
2 from tweepy import OAuthHandler
3 from tweepy.streaming import StreamListener
4 from pymongo import MongoClient
5 import json
6 import sys
7
```

```
8
9
10 MONGO_HOST= 'mongodb://localhost/miscdb'  # assuming you have mongoDB installed locally
11                                          # and a database called 'twitterdb'
12 ckey = "pF5W6BaHFteEYdkq38cgZ755g"
13 csecret = "AwnBHQTmvVr6cMSYKm89vQOXMO3sPyoGvhXMaow3zlg2G74vLq"
14 atoken = "2646419088-gtzEn525GUUtTlwpegdZGd8dGLWl8m2IGq9jk9y"
15 asecret = "iQbg8ZfJ8KkI06n0lwkCVVprtkzsIp6h1RMXP5LN3Z6o2"
16
17 class StreamListener(StreamListener):
18     #This is a class provided by tweepy to access the Twitter Streaming API.
19
20     def on_connect(self):
21         # Called initially to connect to the Streaming API
22         print("You are now connected to the streaming API.")
23
24     def on_error(self, status_code):
25         # On error - if an error occurs, display the error / status code
26         print('An Error has occured: ' + repr(status_code))
27         return False
28
29     def on_data(self, data):
30         #This is the meat of the script...it connects to your mongoDB and stores the
       tweet
31         try:
32             client = MongoClient(MONGO_HOST)
33
34             # Use twitterdb database. If it doesn't exist, it will be created.
35             db = client.miscdb
36
37             # Decode the JSON from Twitter
38             datajson = json.loads(data)
39
40             #grab the 'created_at' data from the Tweet to use for display
41             created_at = datajson['created_at']
42
43             #print out a message to the screen that we have collected a tweet
44             print("Tweet collected at " + str(created_at) + " " + datajson['text'])
45
46             #insert the data into the mongoDB into a collection called twitter_search
47             #if twitter_search doesn't exist, it will be created.
48             db.misc.insert(datajson)
49         except Exception as e:
50            print(e)
51
52
53 auth = OAuthHandler(ckey, csecret)
54 auth.set_access_token(atoken, asecret)
55 #input = input("Which districts would you like to observe? Please separate districts by a
       comma")
56 #keywords = list(input.split(','))
57 #keywords = ["Antonio Delgado", "John Faso", "NY19", "@DelgadoforNY19", "@RepJohnFaso"]
58 keywords = ["#midterms", "#vote", "#bluewave", "#redwave", "#democrats", "#elections", "#
       gop", "#usa"]
59 twitterStream = Stream(auth, StreamListener())
60 twitterStream.filter(track=keywords)
61 '''
62
63 '''
```

Name: Sentiment Profile

Purpose: This code creates a sentiment profile for a given search term and a given database. The sentiment profile is the same profile that's discussed at the end of Chapter 3

Execution: This code was written under the assumption that a mongodb database is set up with data looking for. You'll also need vaderSentiment and Pymongo installed through pip. The script is executed from the command line using the following template:

```
$python3 sentiment_profile.py <politician> <database>
```

Expected Output: The output will be a sentiment profile printed to the console:

```
Search Term: Claire McCaskill Race: Missouri Senate Count: 27940 \newline
Pos Tweets: 23663 Neg Tweets: 2328 Neu Tweets: 1949 Average Compound: 0.392
```

```
import ast
import sys
from configparser import ConfigParser
from pymongo import MongoClient
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

MONGO_HOST = 'mongodb://localhost/'
POLITICIAN = sys.argv[1]
DB = sys.argv[2]

def gather_tweets(phrase, collection):
    cursor = collection.find({"text": {"$regex" : phrase, "$options": "$i"}})
    entries = [entry for entry in cursor]
    return(entries)

def strip_tweet(text):
    text = text.replace('\n', '')
    return(text)

if __name__ == "__main__":
    config = ConfigParser()
    client = MongoClient(MONGO_HOST)
    analyzer = SentimentIntensityAnalyzer()
    if sys.argv[2] == 'house':
        db = client.housedb
        presort = db.presort
    if sys.argv[2] == 'senate':
        db = client.senatedb
        presort = db.presort
    if sys.argv[2] == 'misc':
        db = client.miscdb
        presort = db.misc
    entries = gather_tweets(POLITICIAN, presort)
    ave_score = 0
    pos_tweets = 0
    neg_tweets = 0
    neu_tweets = 0
    count = 0
    for i in entries:
```

```
        tweet = strip_tweet(i['text'])
        sentiment = analyzer.polarity_scores(tweet)
        if (sentiment['compound'] == 0):
            neu_tweets += 1
        if (sentiment['compound'] > 0):
            pos_tweets += 1
        if (sentiment['compound'] < 0):
            neg_tweets += 1
        ave_score += sentiment['compound']
        count += 1
    if count == 0:
        print("No Tweets for this Politician exist in this database")
    else:
        print("Average score:", ave_score/count, "Positive Tweets:", pos_tweets,
              "Negative Tweets:", neg_tweets, "Neutral Tweets:", neu_tweets)
```

Name: Build Search Profile

Purpose: This script takes a list of all the races and search terms and creates a sentiment profile to each one and then writes that sentiment profile to a file. That allows use that file for analysis any which way we like.

Execution: This script needs an accompanying configuration file. That means that you'll also need the Python library ConfigParser to read that configuration file. The other requirements are the same as those required by the Sentiment Profile script. To run the script, use the command:

```
python3 build_search_profile.py senate
```

To build the profile for the house, use house as the input instead of senate with the same command format as above.

Expected Output: The file prints whichever race it's currently working on to the console. The final output will be a .csv file it creates in the working directory.

```
import ast
import sys
from configparser import ConfigParser
from pymongo import MongoClient
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

MONGO_HOST = 'mongodb://localhost/'
FILE_NAME = 'races.ini'
keywords = ["#midterms", "#vote", "#bluewave", "#redwave", "#democrats", "#elections", "#
    gop", "#usa"]
#POLITICIAN = sys.argv[1]
DB = sys.argv[1]

def section_to_dict(section, parser):
    #change path to your ini file if running locally
    parser.read(FILE_NAME)
    out_dict = {}
    for key in parser[section]:
        temp = ast.literal_eval(parser[section][key])
```

```
19          out_dict[temp[-1]] = temp[:-1]
20      return(out_dict)
21
22  def gather_tweets(phrase, collection):
23      cursor = collection.find({"text": {"$regex" : phrase, "$options": "$i"}})
24      entries = [entry for entry in cursor]
25      return(entries)
26
27  def strip_tweet(text):
28      text = text.replace('\n', '')
29      return(text)
30
31  def build_sentiment_profile(politician, db_choice):
32      client = MongoClient(MONGO_HOST)
33      analyzer = SentimentIntensityAnalyzer()
34      if db_choice == 'house':
35          db = client.housedb
36          presort = db.presort
37      if db_choice == 'senate':
38          db = client.senatedb
39          presort = db.presort
40      if db_choice == 'misc':
41          db = client.miscdb
42          presort = db.misc
43      entries = gather_tweets(politician, presort)
44      ave_score = 0
45      pos_tweets = 0
46      neg_tweets = 0
47      neu_tweets = 0
48      count = 0
49      for i in entries:
50          tweet = strip_tweet(i['text'])
51          sentiment = analyzer.polarity_scores(tweet)
52          if (sentiment['compound'] == 0):
53              neu_tweets += 1
54          if (sentiment['compound'] > 0):
55              pos_tweets += 1
56          if (sentiment['compound'] < 0):
57              neg_tweets += 1
58          ave_score += sentiment['compound']
59          count += 1
60      return (ave_score, pos_tweets, neg_tweets, neu_tweets, count)
61
62  if __name__ == "__main__":
63      config = ConfigParser()
64      if DB == 'senate':
65          senate = section_to_dict('SENATE', config)
66          f = open('senate_profile.csv', 'w+')
67          f.write('Search Term,Race,Count,Pos Tweets,Neg Tweets,Neu Tweets,Average Compound
        \n')
68          for i in senate:
69              print(i)
70              for term in senate[i]:
71                  ave_score, pos_tweets, neg_tweets, neu_tweets, count =
        build_sentiment_profile(term, 'senate')
72                  if (count > 0):
73                      line = str(term) + ',' + str(i) + ',' + str(count) + ',' + str(
        pos_tweets) + ',' + str(neg_tweets) + ',' + str(neu_tweets) + ',' + str(ave_score/
        count) + '\n'
74                  else:
75                      line = str(term) + ',' + str(i) + ",0,0,0,0,0\n"
76                  f.write(line)
77      if DB == 'house':
78          senate = section_to_dict('HOUSE', config)
```

```
79          f = open('house_profile.csv', 'w+')
80          f.write('Search Term,Race,Count,Pos Tweets,Neg Tweets,Neu Tweets,Average Compound
    \n')
81          for i in senate:
82              print(i)
83              for term in senate[i]:
84                  ave_score, pos_tweets, neg_tweets, neu_tweets, count =
    build_sentiment_profile(term, 'house')
85                  if (count > 0):
86                      line = str(term) + ',' + str(i) + ',' + str(count) + ',' + str(
    pos_tweets) + ',' + str(neg_tweets) + ',' + str(neu_tweets) + ',' + str(ave_score/
    count) + '\n'
87                  else:
88                      line = str(term) + ',' + str(i) + ",0,0,0,0,0\n"
89                  f.write(line)
90      if DB == 'misc':
91          f = open('misc_profile.csv', 'w+')
92          f.write('Search Term,Count,Pos Tweets,Neg Tweets,Neu Tweets,Average Compound\n')
93          for i in keywords:
94              print(i)
95              ave_score, pos_tweets, neg_tweets, neu_tweets, count =
    build_sentiment_profile(i, 'senate')
96              line = str(i) + ',' + str(count) + ',' + str(pos_tweets) + ',' + str(
    neg_tweets) + ',' + str(neu_tweets) + ',' + str(ave_score/count) + '\n'
97              f.write(line)
98          f.close()
99      f.close()
```

Name: Naive-Bayes Modelling

Purpose: The script models a candidate's probability of winning given their sentiment profile. It reads in the data from a given csv, splits the data into training and testing sets, and then trains the algorithm. The script then predicts a series of outputs given the testing set, and then calculates the error.

Execution: Running this will need the python libraries Sci-Kit Learn or sklearn, pandas, and numpy. It will also need the sentiment profiles that were created from the previous scripts

Expected Output: The script will print out the accuracy of the model to the console. It can be toggled with to print the output as well as the probabilities for the output (bayesian modeling returns a probability of being 1 or 0).

```
1  import pandas as pd
2  import numpy as np
3  from sklearn.naive_bayes import GaussianNB
4  from sklearn.linear_model import LogisticRegression
5
6  FILE_NAME = "senate_profile.csv"
7
8  def accuracy(output, answers):
9      right = 0
10     print("Predicted, Actual, Result")
11     for i in range(len(answers)):
12         if (output[i] == answers[i]):
13             right += 1
```

```
14              print(output[i], answers[i], "Right")
15          else:
16              print(output[i], answers[i], "Wrong")
17      return(float(right/len(answers)))
18
19 senate = pd.read_csv(FILE_NAME)
20 train_data = np.asarray(senate.loc[:70, 'Count':'Average Compound'])
21 train_labels = np.asarray(senate.loc[:70, 'Winner'])
22 test_data = np.asarray(senate.loc[71:, 'Count':'Average Compound'])
23 test_labels = np.asarray(senate.loc[71:, 'Winner'])
24
25 gnb = GaussianNB()
26 gnb.fit(train_data, train_labels)
27 output = gnb.predict(test_data)
28 probs = gnb.predict_proba(test_data)
29 print(senate.loc[71:, 'Search Term':'Winner'])
30 print(output)
31 print(probs)
32 print(accuracy(output, test_labels))
```