# CS440_Project

December 18, 2019

## 1 Sentiment Analysis on Presidential Candidates

### 1.1 CS 440 Final Project

### 1.2 Nicholas Kaufhold

#### 1.2.1 Introduction

For this project we will be utilizing python libraries as well as a database of Tweets that my colleague and friend, Robert Cope, has allowed me to access. The python libraries include nltk, pandas, and other more commonly used libraries such as random and string. In order to retrieve data from the database, I was provided with a method and its parameters to do so. Utilizing Twitter's advanced search he was able to align an advanced search with presidential names I hand selected in which the search stored the tweets that either contain the presidential candidates name or was tweeted by the candidate themselves. This project will explore sentiment analysis tools from the Python nltk library and describe noticeable features I have found in analyzing the outputs.

#### 1.2.2 Twitter Aggregation System

In order to collect enough data to perform an analysis on my colleague Robert Cope provided a method to pull the data from the database he owns and is described as the Twitter aggregation system. The Twitter aggregation system being consumed by your project is relatively simple and may be considered from a high level: a user gives the system one or more Twitter "advanced search" queries. These queries are loaded by a batch job once every 4-6 hours, and the Twitter API is polled for new tweets. Any tweets not currently present in the database are added to the database while this batch job runs; the schema utilized is essentially the same as Twitter's own data schema (i.e. users, tweets and "media items"). The other major component is a set of Python database query wrappers, which wrap relatively simple select statements, and load back matching tweets and users as record objects. For this project, we're using Postgres full text search primitives (SIMILAR TO syntax) to find matching tweets to examine, in addition to looking for specific twitter users tweets. (Note: If it is at all desired to further understand how the database works, please feel free to contact Robert Cope at rpcope1@gmail.com.)

#### 1.2.3 Creating a sufficient model to train tweets on

Based on my research, I was able to conclude that Python's nltk library is powerful and versatile enough in order to properly create a model that can train and test on tweets, or in my case, tokens [1]. Within this library I was able to download positive and negative tweets, 5000 of each, in order to set up a model that would perform sufficiently on the tweet from the database. In order to perform an analysis on the tweets some important steps were required to prep the data.

One of the main ideas of creating a model is to 'tokenize' data meaning to extract each word as its own token. Once a word is taken out of a tweet is is important to give it context. This is done through a process called lemmatization [2]. This will chop any word down to its canonical form for example, 'running' to 'run' or 'months' to 'month'. Below is an example of a tokenized tweet and then a tokenized tweet that has been cleaned up through lemmatization:

[ ]:
```
positive_tweet_tokens = twitter_samples.tokenized('positive_tweets.json')
negative_tweet_tokens = twitter_samples.tokenized('negative_tweets.json')
```

Let's see what the first positive tweet tokens look like.

[41]:
```
print(negative_tweet_tokens[1])
print(negative_tweet_tokens[3])
```

```
['Everything', 'in', 'the', 'kids', 'section', 'of', 'IKEA', 'is', 'so', 'cute',
'.', 'Shame', "I'm", 'nearly', '19', 'in', '2', 'months', ':(']
['"', '@ketchBurning', ':', 'I', 'hate', 'Japanese', 'call', 'him', '"', 'bani',
'"', ':(', ':(', '"', 'Me', 'too']
```

[42]:
```
positive_cleaned_tokens_list = []
negative_cleaned_tokens_list = []

for tokens in positive_tweet_tokens:
    positive_cleaned_tokens_list.append(clean_tweet(tokens))

for tokens in negative_tweet_tokens:
    negative_cleaned_tokens_list.append(clean_tweet(tokens))

print(negative_cleaned_tokens_list[1])
print(negative_cleaned_tokens_list[3])
```

```
['everything', 'kid', 'section', 'ikea', 'cute', 'shame', "i'm", 'nearly', '19',
'2', 'month', ':(']
['"', 'hate', 'japanese', 'call', 'ban', ':(', ':(', '"']
```

We did a few important things here. In the clean_tweet() function we take the tokens fron our list of tweet tokens and identify what type context each word is using the pos_tag() function. This function identifies whether a word is a proper noun, a common noun, a verb (past or present), etc. In order to properly lemmatize the tokens, it is necessary to tag the tokens with 'n' for nouns, 'v' for a verb, or 'a' for adjective. Then when we pass the token and its associated tag to the lemmatizer, it will determine which cononical form of the word to use. Finally we pass the tokens through a logical statement in which we determine if the token is puncuation or a stop word. And above is the final product. Notice that all words are now lowercase, '@ketchBurning' is removed, stop words such as 'in', 'is', and 'me' are removed, and 'kids' and 'months' have been converted to 'kid' and 'month'.

Next we would like to set up our data for the model. Per the documentation the tweet dataset has 5000 tweets for both the positive tweet and negative tweet dataset [3]. So we will initialize each tweet as a dictionary using what I call cleaned tweets and then set the key to positive or negative

for each dataset. We then shuffle the data so that we can train the data. After some experimenting I found I could use 6500 tweets for the training dataset and as we will see, get a high accuracy when testing the data.

[268]:
```
positive_tokens_dict = cleaned_tweets_dict(positive_cleaned_tokens_list)
negative_tokens_dict = cleaned_tweets_dict(negative_cleaned_tokens_list)

positive_tweet_dataset = [(positive_tweet_dict, "Positive") for␣
↪positive_tweet_dict in positive_tokens_dict]
negative_tweet_dataset = [(negative_tweet_dict, "Negative") for␣
↪negative_tweet_dict in negative_tokens_dict]

tweet_dataset = positive_tweet_dataset + negative_tweet_dataset

random.shuffle(tweet_dataset)

train_data = tweet_dataset[:6500]
test_data = tweet_dataset[6500:]
```

We can take a look at the most common negative tokens. Notice the frowning emoticons are the most common.

[55]:
```
positive_words = get_words(negative_cleaned_tokens_list)
freq_dist_positive = FreqDist(positive_words)
print(freq_dist_positive.most_common(10))
```

```
[(':(', 4585), (':-(', 501), ("i'm", 343), ('…', 332), ('get', 325), ('miss',
291), ('go', 275), ('please', 275), ('want', 246), ('like', 218)]
```

I decided to use the Naive Bayes algorithm in order to train the data. The nltk library has available the NaiveBayesClassifier.train() method which is very simple to use. Thus, I pass in the training data and the classifier will consider all tokens in a tweet and based on each token value (positive or negative) it will assign the tweet as a whole a positive or negative. Once I did this I found that I had an accuracy of 99.7%!

[59]:
```
classifier = NaiveBayesClassifier.train(train_data)
print("Training data accuracy:", classify.accuracy(classifier, test_data))
```

```
Training data accuracy: 0.9968571428571429
```

I feel that this will be reasonable to utilize to run on the presidential tweets. Some of the most informative features (this is a method of the classifier class in the nltk library) showed the following statistics:

[60]:
```
classifier.show_most_informative_features(15)
```

```
Most Informative Features
                      :( = True           Negati : Positi =   1886.9 : 1.0
                      :) = True           Positi : Negati =   1529.4 : 1.0
                follower = True           Positi : Negati =     33.1 : 1.0
```

```
                  sad = True           Negati : Positi =     22.2 : 1.0
                 sick = True           Negati : Positi =     17.9 : 1.0
            community = True           Positi : Negati =     17.4 : 1.0
                  ice = True           Negati : Positi =     15.3 : 1.0
                 miss = True           Negati : Positi =     15.3 : 1.0
             followed = True           Negati : Positi =     13.9 : 1.0
               arrive = True           Positi : Negati =     12.4 : 1.0
                  ugh = True           Negati : Positi =     12.1 : 1.0
                 glad = True           Positi : Negati =     11.7 : 1.0
           definitely = True           Positi : Negati =     11.3 : 1.0
              welcome = True           Positi : Negati =     10.3 : 1.0
        unfortunately = True           Negati : Positi =     10.1 : 1.0
```

What this is saying is that in general the ':(' emoticon appears about 1887 times more in negative tweets than it does in positive tweets. So what we've essentially done is set up our model such that for every token in a tweet we submit to the model it will find the same token and base its inference on positive or negative based on these ratios. So if a tweet has the word 'glad', since the model saw glad in more positive tweets than negative tweets, it will assign glad a value of positive.

[61]:
```
custom_tweet_negative = "@USSenate: This is terrible. All the candidates want␣
↪to raise taxes."
custom_tokens_negative = clean_tweet(word_tokenize(custom_tweet_negative))
print(classifier.classify(dict([token, True] for token in␣
↪custom_tokens_negative)))
```

```
Negative
```

Unfortunately we are relying on the fact that most negative tweets will not contain positive tokens. If we simply add in some positive tokens into a tweet, we can change the outcome.

[63]:
```
custom_tweet_ambiguous = "@USSenate: This is gladly terrible. All the␣
↪candidates want to happily raise taxes."
custom_tokens_ambiguous = clean_tweet(word_tokenize(custom_tweet_ambiguous))
print(classifier.classify(dict([token, True] for token in␣
↪custom_tokens_ambiguous)))
```

```
Positive
```

For the sake of this project though, we are going to use our model we created because in my opinion, the tweets we will be analyzing will not have ambiguity such as the tweet above.

### 1.2.4 Calculations on presidential candidate tweets

When initially crafting what data I was going to pull into the database, I looked into the current election candidates [4]. I decided that right away I was not going to include the current president in the results as I wanted to observe who in the current candidate pool has the highest chance at winning the race. I also though that due to the current president's volume of tweets and tweets containing their Twitter handle, this would certainly be a good idea. Thus, I selected the following candidates:

1) Bernie Sanders
2) Andrew Yang
3) Joe Biden
4) Cory Booker
5) Pete Buttigieg
6) Amy Klobuchar
7) Tulsi Gabbard (was selected before she pulled out of the race)
8) Elizabeth Warren
9) Marianne Williamson
10) Julian Castro
11) Tom Steyer
12) Mike Bloomberg

I created a function that pulled the tweet data from the database, gave the tweet a classification, and placed the candidate name, total number of positive tweets, total number of negative tweets, and the sum of the two into a pandas dataframe. I then proceeded to look at each result and determined that probably the most interesting result was the number of positive tweets. If a candidate is to be selected in a poll, surely the person selecting that candidate would have a positive opinion of the candidate.

[122]:
```
df
```

[122]:

| | Candidate | Positive Tweets | Negative Tweets | Total |
|---|---|---|---|---|
| 0 | (B\|b)ernie (S\|s)anders | 160241 | 66673 | 226914 |
| 1 | (A\|a)ndrew (Y\|y)ang | 66542 | 5380 | 71922 |
| 2 | (J\|j)oe (B\|b)iden | 139419 | 386558 | 525977 |
| 3 | (C\|c)ory (B\|b)ooker | 1978 | 638 | 2616 |
| 4 | (P\|p)ete (B\|b)uttigieg | 80654 | 30777 | 111431 |
| 5 | (A\|a)my (K\|k)lobuchar | 10927 | 4472 | 15399 |
| 6 | (T\|t)ulsi (G\|g)abbard | 11545 | 9668 | 21213 |
| 7 | (E\|e)lizabeth (W\|w)arren | 112406 | 95353 | 207759 |
| 8 | (M\|m)arianne (W\|w)illiamson | 480 | 1187 | 1667 |
| 9 | (J\|j)ulian (C\|c)astro | 1607 | 203 | 1810 |
| 10 | (T\|t)om (S\|s)teyer | 2324 | 167 | 2491 |
| 11 | (M\|m)ike (B\|b)loomberg | 10959 | 4681 | 15640 |

Now we will do a little bit of cleaning as well as rearranging of the data:

[270]:
```
df_total_positive
```

[270]:

| Candidate | Percentage Positive |
|---|---|
| Amy Klobuchar | 1.823957 |
| Andrew Yang | 11.107328 |
| Bernie Sanders | 26.747757 |
| Cory Booker | 0.330172 |
| Elizabeth Warren | 18.763041 |
| Joe Biden | 23.272106 |

| | |
|---|---|
| Julian Castro | 0.268244 |
| Marianne Williamson | 0.080123 |
| Mike Bloomberg | 1.829299 |
| Pete Buttigieg | 13.462932 |
| Tom Steyer | 0.387927 |
| Tulsi Gabbard | 1.927115 |

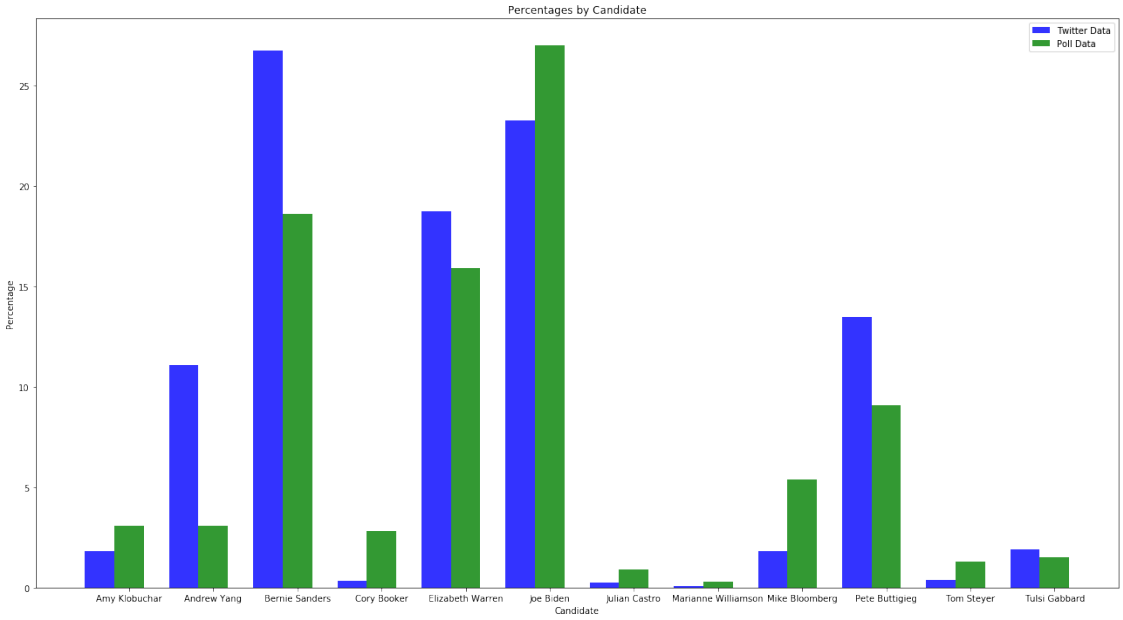
[271]:
```
poll_df_copy
```

[271]:

| Candidate | Poll Percentage |
|---|---|
| Amy Klobuchar | 3.1 |
| Andrew Yang | 3.1 |
| Bernie Sanders | 18.6 |
| Cory Booker | 2.8 |
| Elizabeth Warren | 15.9 |
| Joe Biden | 27.0 |
| Julian Castro | 0.9 |
| Marianne Williamson | 0.3 |
| Mike Bloomberg | 5.4 |
| Pete Buttigieg | 9.1 |
| Tom Steyer | 1.3 |
| Tulsi Gabbard | 1.5 |

The resulting bar chart displays similar percentages for both the twitter positive percentage as well as the poll percentages. It's interesting that both bars appear to have similar magnitudes for the two datasets.

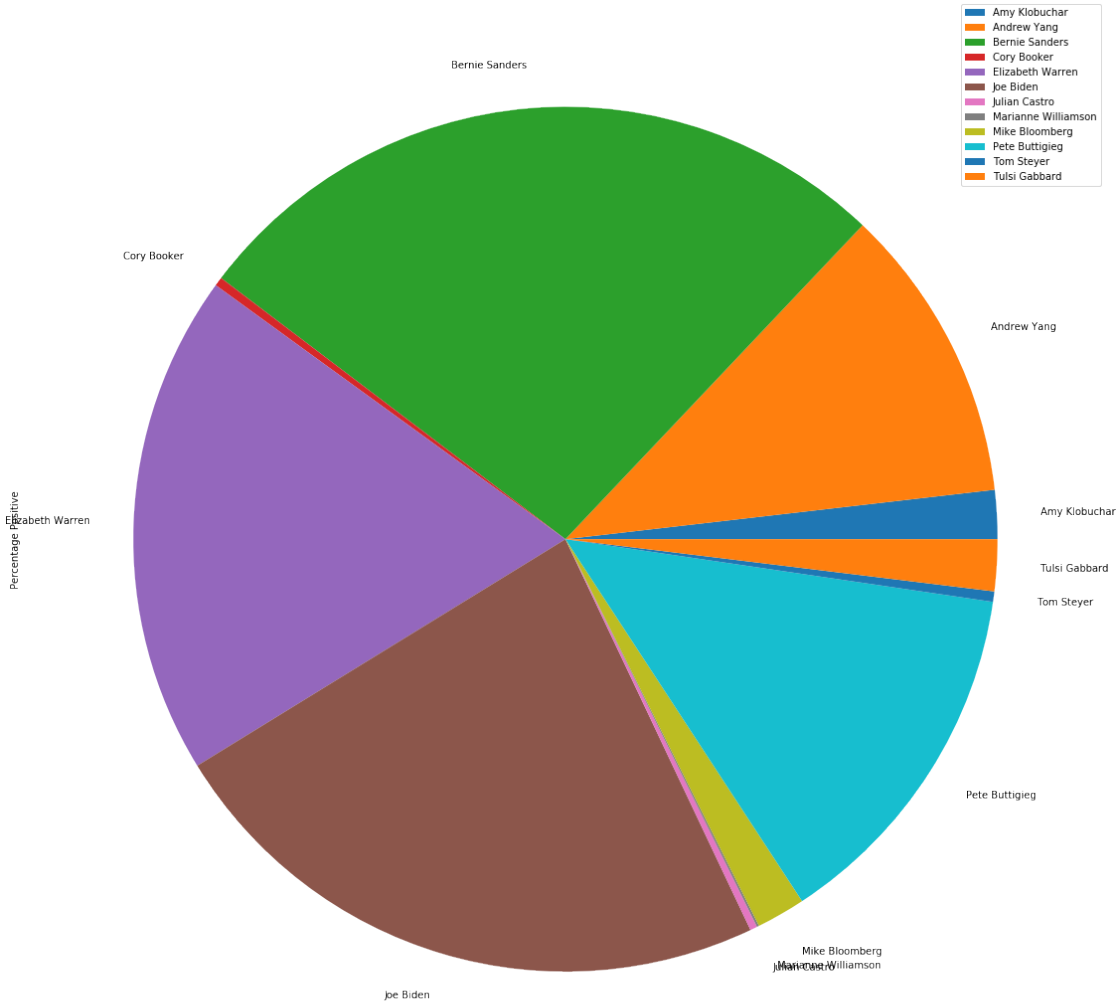
[273]:
```
plot()
```



I chopped down the dataframe into just the candidate name and the positive percentage. Then, utilizing the poll data from [4], I created another dataframe with the same labels. I was able to generate the following pie charts that I feel demonstrate a clear comparison between the positive twitter tweets and poll results:

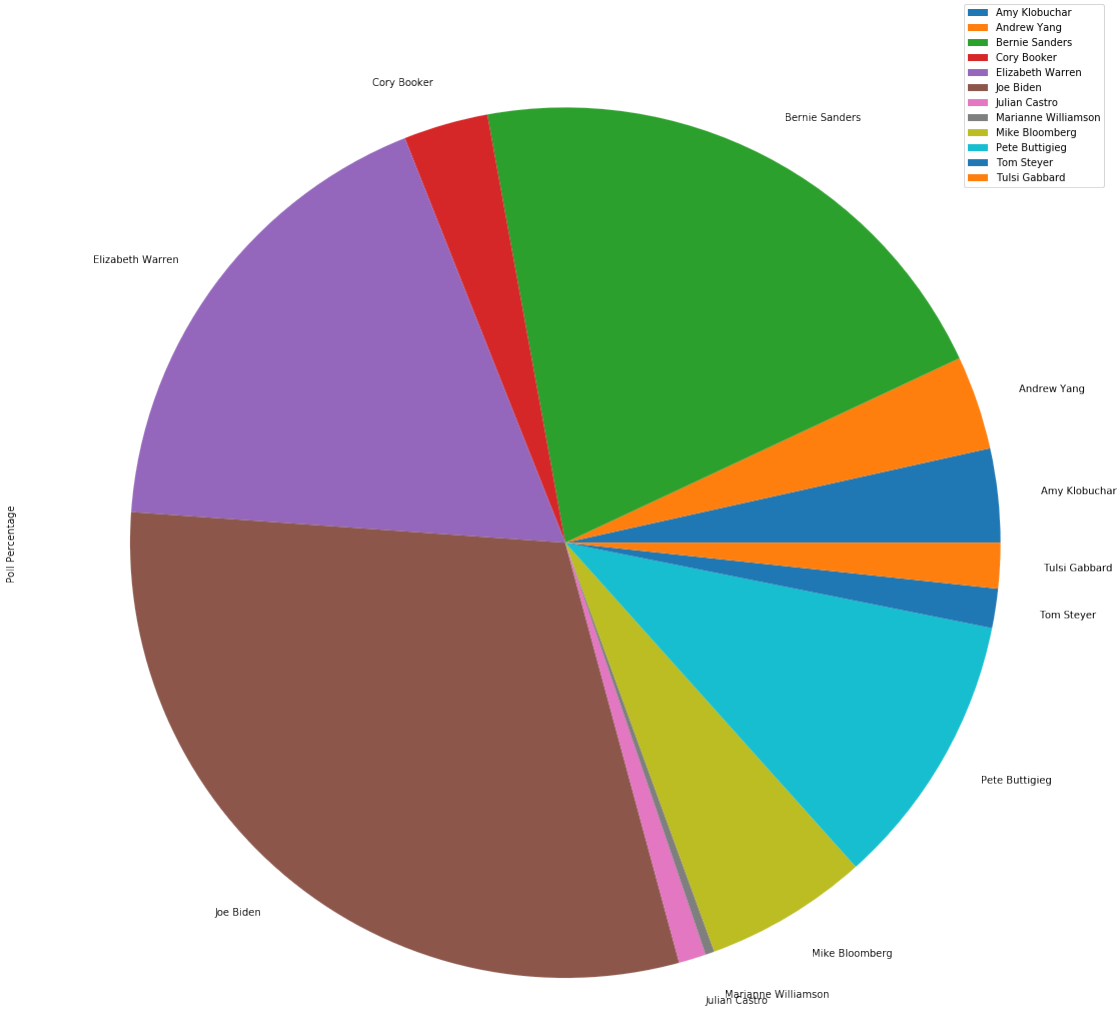
[259]:
```
df_total_positive.plot.pie( y = 'Percentage Positive', figsize = (20,20))
```

[259]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fad86582518>
```




[260]:
```
poll_df_copy.plot.pie( y = 'Poll Percentage', figsize = (20,20))
```

[260]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fad85fd7cc0>
```




### 1.2.5 Dicussion of Results

With a few exceptions such as Andrew Yang, who has a much higher twitter percentage, and Cory Booker, who actually has a higher poll percentage, it does appear that the twitter percentages mock the poll ratings in a way I'd say is almost remarkable! We definitely see some strong support when it's visualized like this.

Some future work I think would be interesting is creating a histogram of the positive and negative tweets per day for each candidate and look at headlines of that day. I'm willing to bet that the sentiment of tweets will alter based on the news that comes out in regard to any candidate.

Finally, the model used has some flaws. As mentioned before, one can mislead the model into thinking a tweet is positive when really it is negative. If people started including tokens that are positive in negative tweets, that can certainly skew the results. Another flaw is that we trained only on the data provided in the nltk library. Perhaps it'd be even better if we trained exclusively

on data pertaining to the presidential election. This way the vocabularies would possibly aligne better.

### 1.2.6 Conclusion

Overall, I believe it has been proven that we showed tweet sentiments do tend to reflect major events such as a presidential election. There is clearly a correlation between the majorities of percentages. Sentiment analysis can certainly be useful whenever there is a plethora of data such as tweets on a Twitter database. This is a sure fire way of describing what a populous may believe to be important. Thank you for reading I hope you enjoyed!

### 1.2.7 References

[1]http://www.nltk.org/nltk_data/

[2]https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

[3]http://www.nltk.org/howto/corpus.html

[4]https://www.realclearpolitics.com/epolls/2020/president/us/2020_democratic_presidential_nomination-6730.html#polls