PolitiTweetStorm

Exploring the Implications of Sentiment Analysis of Political Tweets

Danielle Livneh, Aren Vogel, John Moon, Mathew Wiesman
Wednesday April 20th, 2016

Abstract

The political climate is constantly changing and the issues of today may be obsolete tomorrow. Polititweetstorm aims to provide users dynamic information about the current top political issues. It uses twitter, a dynamic social media site, to determine the top political issues and evaluate the public's sentiment towards these issues in user specified locations. Other sources of political news are typically bias, generalized, or only pertain to a specific population. Polititweetstorm assesses the general sentiment in a region by analyzing what the actual people of that area have to say. It improves upon current polling methods because it removes bias introduced in the polling process by evaluating unfiltered tweets.

This document outlines our final project, PolitiTweetStorm. We have included an updated project description, related work with respective references, a description of our data collection process, results, evaluations, and considerations for the future.

1 Project Description

PolitiTweetStorm is an interactive application that allows users to view how a particular region feels about various current political topics. The user is first prompted to enter a location in the United States. Next, they are shown a list of popular political terms from which they may select. Given a term, our program returns whether the respective region feels positively or negatively about the topic selected. The user is presented with 10 examples positive tweets and 10 example negative tweets from the region, as well as a visual breakdown of the distribution of sentiments. Ultimately PolitiTweetStorm is a region specific political sentiment analysis tool.

We believe that by providing compelling data on the present political issues of a region users will be able to obtain an unbiased representation of constituent political attitudes. Given the sensationalistic nature of large media today we think this is not only exciting but also important to users sand our nation alike.

PolitiTweetStorm can broken up into three main sections:

1. Finding the most popular relevant political terms
2. Collecting all tweets of a given region containing a specific political term
3. Using sentiment analysis to determine the tweets collected in step two as positive or negative

Step one involves crawling specific political twitter accounts to obtain a dictionary of terms. We take the top terms and offer these to our user as options for search terms. To accomplish this we use Twitter's API and the Twitter Python library, as well as common preprocessing techniques (removing stopwords, punctuation, tokenizing etc.)

Step two takes the user's input (term and geographic region) as query parameters and, using twitter's API, returns all possible tweets given the restrictions Twitter has on to how many tweets we have access.

The next part uses Naïve Bayes to analyze the sentiment of these tweets. Prior to running our application we trained on over 1 million tweets from a dataset we found online. We used the frequencies from this dataset to conduct our sentiment analysis on the queried tweets.

2 Related Work

Considering the specificity of the topic, we were surprised to learn that there already exists a lot of research on sentiment analysis of political tweets. The first source (mentioned in lecture), "NRC-Canada: Building the State-of-the-Art in Sentiment Analysis of Tweets" (Mohammad et al.), focused mostly on building sentiment analysis for tweets. The methods they mentioned were helpful in spawning ideas of how we wanted to go forward with our project. We were able to adapt their idea of collecting seed words to look for in tweets to work better for us. Along with looking for positive and negative sentiment we also needed another seed set of political terms.

This paper also provided a lot of ideas how we will want to score the overall sentiment of tweets. The paper mentions a few methods one of which is "score(w) = PMI(w, positive) − PMI(w, negative) (1) where PMI stands for point wise mutual information." (Mohammad et al.). We can adapt this to a simple scoring mechanism that just checks the number of words that are positive vs the number of words that are negative to make our choice.

The next source, "Analyzing Political Sentiment on Twitter" (Ringsquandl et al.), provided us with a strong methodology for sentiment analysis that had specifically been done with politics on Twitter in the US. Along with that it will provide us with a good expected measure of the results of our system. In their conclusion they mention that Twitter needs special consideration to how it is processed. This made us aware that how we handle twitter-talk will be much different than a more standard document.

The third paper we looked at, “Towards Tracking Political Sentiment through Microblog Data” (Wang et al.), brought to our attention the importance and usefulness of twitter data when dealing with politics. “We consider Twitter as important example of social expression of opinion...In the context of politics, Twitter con- tent, together with Twitter users’ 88 information, such as user’s profile and social network, have shown reasonable power of detecting user’s politi- cal leaning (Conover et al., 2011) and predicting elections (Tumasjan et al., 2010).”(Wang et al.). On top of that, their mentioned methods of using the Twitter API to track specific topics were use- ful. They also used Naive Bayes as their method of classification, which gave us good insight that al- though it is Naive it usually works quite well as may be a good classification method for us to use.

"From Tweets to Polls: Linking Text Sentiment to Public Opinion Time Series." brought to our attention that a system like our could be used as an alternative to polling. Not only is it much cheaper, “mining public opinion from freely available text content could be a faster and less expensive alternative to traditional polls.” (O’Connor et al.) but “[s]uch analysis would also permit us to consider a greater variety of polling questions, limited only by the scope of topics and opinions people broadcast” (O’Connor et al.) The paper’s discussion on Text Analysis and break down into two steps of “message retrieval: identify messages relating to the topic.” and “opinion estimation: determine whether these messages express positive or negative opinions or news about the topic.” (O’Connor et al.) is exactly how we aimed to design our system. Both their positive discussion of the correlation of polling and twitter as an indicator as well as their discussion of using twitter as a way to predict or “forecast” polls (even with a simple sentiment analysis system) gave us a good outlook on our system as something that could have a useful short term and long term purpose.

“Classifying Political Orientation on Twitter: It’s Not Easy!” (Cohen et al.) brought a bit of light to the idea that our system/project could be perfect. Many of the other papers mention success with decently strong results while this paper looks deeper into those to find that often those values were over “optimistic”. In reality, classifying po- litical leanings doesn’t always work in every con- text. They point out a few key parts that are tough, namely “Good, labeled datasets are hard to build” and “Existing methods fail on “ordinary” users.” (Cohen et al.). Both of these are good warnings, letting us know that our system may not be as ro- bust as we may have originally planned it.

"A System for Real-time Twitter Sentiment Analysis of 2012 U.S. Presidential Election Cycle." helped us to design a real time system as described in the paper by Wang et al. The paper describes a system architecture in Figure 1. We were able to adapt this to model to fit our project scheme. For example, where they discuss matching a tweet to be about a certain candidate we look for a specific trending topic (which could be a candidate). The paper also mentions using twitter specific tokenizers which brought their existence to our attention as a possibility for something we could use. Their front end focuses on showing statistics and data. This brought to our attention that we wanted something that was easy for all users to understand, not just people who wanted analytics. We also took into account various other display and design choices they used when working on our front end. “The architecture and method are generic, and can be easily adopted and extended to other domains.” (Wang et al.) Not only were we able to adopt and extend their methods we also aimed to build our system in such a way that it could also be adopted and extended for other uses.

All of these sources allowed us to have a better understanding of the scope of our project and how to best approach it with certain ideas in our heads.

3 Data Collection

In order to determine the current trending political issues on twitter we used the twitter API to crawl and obtain tweets from four prominent political news accounts. We pulled tweets from: @NPRPolitics, @ HuffPostPol, @NBCPolitics, and @CNNPolitics. After obtaining all the tweets we removed stopwords, links, non-alphanumeric characters, and twitter frequent terms.

The following is an example list of the top 20 terms with their word counts produced from our program run on March 25th, 2016:

('trump', 119) ('obama', 76) ('donald', 55) ('brussels', 48) ('clinton', 45) ('president', 43) ('cuba', 39) ('attacks', 38) ('cruz', 34) ('finalfive', 33) ('sanders', 29) ('gop', 26) ('hillary', 26) ('court', 23) ('ted', 20) ('bernie', 19) ('supreme', 18) ('watch', 17) ('kasich', 17) ('havana', 17)

When our program is run we create this list in real time. In order to create our set of “Top Political Issues” we loop through the list of top terms we return and check for any common bigrams. For example in our list we return Donald and Trump as two separate terms. After we loop through the terms as bigrams we conclude that Donald and Trump really should be one term of “Donald Trump”.

The following is an example list of the top 20 bigrams and their respective counts from our program run on April 19th, 2016:
('new', 'york'):45 ('donald', 'trump':)35 ('hillary', 'clinton'):25 ('bernie', 'sanders'):21 ('on', 'the'):19 ('in', 'new'):15 ('the', 'nyprimary'):15 ('for', 'the'):13 ('in', 'the'):12 ('win', 'the'):11 ('9', '11'):10 ('the', 'gop'):10 ('the', 'new'):10 ('at', 'the'):10 ('of', 'the'):9 ('u', 's'):9 ('cnn', 'projects'):9 ('it', 's'):9 ('tonight', 's'):8 ('clinton', 's'):8

Referenced these bigrams with our list complementary list of top twenty unigrams, our system produces the following as the top trending political topics:
['new york', 'donald trump', 'hillary clinton', 'bernie sanders', '9 11', 'nyprimary', 'primary']

Once we have these bigrams and unigrams we are able to use the Twitter API to pull tweets about each of these topics for a user specified region. The Twitter API does a simple keyword search and returns all tweets in the specified region with the specific political topic. Because of the restrictions placed on us by the Twitter API, we are only able to grab around 100 tweets or less from the region given the political term.

Once we query these tweets, we run them through our Naïve Bayes system, and classify each tweet as either negative or positive. If the majority of tweets are positive we declare that specific region to feel positively about the respective topic, and vice versa. This ultimately allows us to print our final data of how for or against a specific region is on top political issues.

The dataset we used to train our Naïve Bayes system is called the Twitter Sentiment Analysis Datatset. According to ThinkNook.com, this dataset is based on data from University of Michigan's Sentiment Analysis competition on Kaggle and the Twitter Sentiment Corpus by Niek Sanders. This dataset contains over 1,578,627 classified tweets, with 1 denoting positive sentiment and 0 indicating negative sentiment. We threw out a number of tweets that contained characters our system could not interpret. The tweets are not diverse, and generally not political in nature, which certainly impacted our ability to accurately analyze the sentiment of our political tweets. We downloaded the csv file linked to on http://thinknook.com/twitter-sentiment-analysis-training-corpus-dataset-2012-09-22/ and stored it locally for training. Some examples from this training set include:

- "I wanted to sleep in this morning but a mean kid through a popsicle stick at me head. I wish I could fly away like those squirrels" (negative)
- "had a Really good night!!! Stoked for things to come!!!"(positive)
- "Dark Knight stiil good after the 5th time." (positive)
- "oh no my dog just attacked the cat belongong to my relatives now he's in the doghouse! It's the kinda thing that rips families apart.."(negative)

4 Method Descriptions and Evaluation

The main feature of our application is the sentiment analysis of the retrieved tweets as the Twitter API largely takes care of returning the tweets we want to classify. Our Naïve Bayes implementation assumes the "bag of words" model, where the probability of each word is independent from the other words.

For each line in the training data, we parsed the tweet text and the subsequent sentiment. We applied a basic NLTK tokenizer on the tweets, but kept stop words as we believe they are important to sentiment.
We applied the same preprocessing on our incoming tweets.

Although the training data we used for the live application uses all possible tweets, we evaluated our system by training on a subset of the dataset, and then testing on the remaining tweets.

We define our systems accuracy to be how many tweets it correctly classifies out of the total number of tweets.

We ran two separate tests: once by leaving out 1000 tweets from the training dataset and once leaving out 100,000 tweets from the training dataset. Out of the 1000 tweets, 765 were correctly classified (76.5% accuracy. Similarly 76,309 tweets out of the 100,000 tweets were correctly classified (76.3%). This suggests our Naïve Bayes classifier is approximately 76% accurate.

We manually tested how accurate our system was for a subset of retrieved tweets. Out of the 85 Bernie Sanders tweets from Bethesda, Maryland our system accurately classified 48 tweets (56.7%). We believe this discrepancy is due to the political nature of our tweets and nonpolitical nature of our training set, as well as our tokenizing process. When we switched to an NLTK TwitterTokenizer to the incoming tweets and tested the same 85 tweets, we got an accuracy of 64%. We hypothesize with refined tokenizing and a more diverse training set we could greatly increase the accuracy of our system even more.

5 Individual Contributions

Aren:

I specifically worked with dynamically generating the top political topics, processing the user's query, and retrieving all relevant tweets for a search.

In order to generate the top political terms I used twitter's API to pull the recent tweets from CNN, Huffington Post, NPR, and NBC's political twitter accounts. Then, I parsed the returned JSON objects, tokenized the tweets, removed stop words, and computed bigram and unigram frequencies. After generating the top terms and bigrams we cross-referenced the lists for bigrams that had both terms on the top unigram list. This allowed us to create better-defined terms like, "Bernie Sanders" rather than "Bernie" and "Sanders" as separate topics.

After we dynamically generate the top political topics, we present the topics to the user and allow them to specify a location. I incorporated the Google maps API and included the search bar to auto complete the user's location query. When the search is entered, we use Google Map's API to request the longitude and latitude of the user's specified location, to have for the twitter requests. When that is returned we retrieve all the relevant tweets for the user specified topic and geocoordinates and generate the necessary data structures for sentiment analysis.

John:

Most of my work revolved around setting up the website that hosted our live system. To easily enable the connection between our front end visualization and backend code, which was mostly written in Python, I decided to use the Flask framework and utilized Jinja to render our templates. On each local environment, we downloaded various Python packages, detailed in requirements.txt - for the frontend, we downloaded Flask, Jinja 2.0, and Gunicorn, which creates a local server for a Flask website; for the backend, we downloaded NLTK, Pandas, Twitter, and Google Maps. I researched the various API's we decided to use, and helped Aren get started in terms of using our data to retrieve accurate information from Twitter and Google Maps. In the future, I'm hoping to implement the heat map that one of the other groups had, which would display the density of tweets in a region, indicating positive or negative sentiment. In addition, I used Chart.js to display the results of our system in an aesthetically pleasing graph. For the design of our website, I used Materialize.css to style various elements not possible in standard HTML or Bootstrap.

Mathew:

A good portion of my work began in researching what did and didn't work when doing sentiment analysis in the field of politics. I used both Google Scholars and the ACL anthology to find relevant articles ranging from a broader scope (sentiment analysis of a given topic) to much more narrow articles (specifically sentiment analysis of politics on twitter).

From there I worked with my teammates to help design the overall schema we would use to create the system. More specifically I worked on creating the classification system. The first step was to create a training system. We aimed to find a large dataset of already classified (as positive or negative) tweets. Once the ultimately used CSV file was acquired, Dani and I built a training system based off the Naive Bayes formula and word frequency towards sentiment. After noticing that running training on such a large dataset would be very time consuming we suggested that we store the resulting trained system (python dictionaries) as a JSON file for easy loading when needed in running classification.

The next step was to build the actual classification system that would be used to classify incoming tweets relevant to the chosen topic and location. Here I also worked with Dani to create a Naive Bayes classifier. The system would take in the tweets and tokenize them. Then based on the log conditional probabilities the overall sentiment probability for both positive and negative were determined. The higher probability was the sentiment that was chosen. Other relevant data (such as example Positive and Negative tweets and the ratio of positive to negative tweets in the area) was also calculated and return to the controller to be displayed by the resulting view.

I also worked with John in setting up the Python Flask application, more specifically on the side of setting up the controllers and the connection between them and the template view.

Dani:

The majority of my work lied in establishing our sentiment analysis processes. Our first instinct was to use a classifier from the NLTK library, as we hypothesized it would yield higher accuracy than Naïve Bayes. After Mat and I realized that using the NLTK classifier was infeasible given our time and memory constraints, however, we wrote a python program that implemented a traditional Naïve Bayes classification.

The first step was training our system, which we achieved by calculating the probability of each word given each class (positive and negative in this case). We also wrote the code to classify the received tweets once we had the information from the training data stored in JSON objects.

I also worked to gain a better understanding of the accuracy of our system by writing and running the code to test 1000 tweets and 10,000 tweets from the training document. To accomplish this I left out the first 1000 or 10,000 tweets from the training data and constructed the respective frequency dictionaries. Once this step was complete, I ran our system on the

left out tweets while keeping track of how many tweets our program accurately classified.

Finally I assisted John in the CSS design, particularly on the landing page, and organized our reports.

6 Conclusion

Overall our system did a good job of identifying the most discussed political topics and analyzing sentiment of those topics in specified regions. The greatest contribution our application has is with approximately 75% accuracy it can identify the sentiment of a tweet. When this is applied to all tweets containing the selected topic, in a relatively large region, our system is able to give a user a good understanding of how the area feels about the topic. Although it is certainly not perfect, we believe it is a better representation of sentiment than traditional polling.

Some things that worked really well in our implementation are: our methodology for identifying the top political terms, using a large training set of tweets for sentiment analysis, twitter specific tokenization, presenting our application and results in an attractive and user friendly frontend, and categorizing sentiment with the Naïve Bayes formula. The manner in which we created the top political terms was especially creative because it truly creates a dynamic list of political terms without the cost of scrapping tons of twitter data. Other implementations typically rely on less dynamic sources or computationally heavy algorithms, but because we pull the top terms from an array of highly active political news accounts we are able to obtain a set of present day subjects in a matter of seconds.

Our use of a large twitter training set allowed us to specialize our system for twitter sentiment analysis. In development, we realized how different twitter language is from standard dialect. By using twitter specific data sets we increased the likelihood of observing a positively or negatively categorized term when performing real time sentiment analysis. Similarly, by changing our tokenizer to NLTK's TwitterTokenizer our tokens were more twitter specific. Lastly, we did a good job of making our application user friendly and aesthetically appealing. Our use of materialize.css, Google map's API, and the graphical representation of our data made using our system enjoyable and easy to understand.

Like any system, we certainly had our fair share of issues along the way. Some of the core problems we identified were: Twitter API limitations, irrelevant tweets, and deeper sentiment categorization. The Twitter API limited us in a few fashions. First, it limits the number of tweets you are able to pull for a query; Search is rate limited at 180 queries per 15 minute window. They also limit how many tweets are returned for each query and how long ago the tweets can be from. This made it difficult to accurately detect the top political topics, and grab every tweet pertaining to a subject in a certain location. These issues could eventually be overcome by building a sophisticated crawler but with our time frame this was not possible.

Another large problem is twitter does not identify the location of all tweets. Rather, users have the option of specifying their location. Since many users don't identify their location, the number of tweets we are able to pull from a location is significantly lower than the actual number of tweets coming from that area.

Another major problem with our system is that when we pull all "relevant" tweets for a specific topic we inevitably pull some irrelevant tweets. For example in one query for Bernie Sanders we received, "Happy Birthday to my big sis Bernie!! @ Beechtree Homes https://t.co/sZtuWJIEtU". This issue cannot directly be handled but if we are able to increase the number of tweets we receive from each query the impacts on our results will be less significant.

Lastly, we ran into a known large problem of sentiment analysis, issues with deeper sentiment categorization. Our system categorized the tweet as a whole as positive or negative rather than topic specific analysis. For example, a tweet returned for a search about Trump that is "I love Kasich, he is the best candidate around, screw Trump" would be categorized as positive when really it should be identified as negative for a query about Trump. In order to handle this you would have to specifically analyze the sentiment surrounding specific nouns. Although our system does not implement this now, using machine-learning techniques this is certainly something that could be addressed in the future.

Acknowledgments

Shout out to Dr. Mihalcea, Shibamouli, and all our fans out there.

References

Cohen, Raviv, and Derek Ruths. "Classifying Po litical Orientation on Twitter: It's Not Easy!" *School Of Computer Science - McGill University* (n.d.): n. pag. *ACL Anthology*. Web.

Mohammad, Saif M., Svetlana Kiritchenko, and Xiaodan Zhu. "NRC-Canada: Building the State-of-the-Art in Sentiment Analysis of

Tweets." *National Research Council Canada*(n.d.): n. pag. *ACL Anthology*. Web.

O'Connor, Brendan, Ramnath Balasubramanyan†, Bryan R. Routledge, and Noah A. Smith. "From Tweets to Polls: Linking Text Sentiment to Public Opinion Time Series." (n.d.): n. pag. Fourth International AAAI Conference on Weblogs and Social Media. Web. <https://www.aaai.org/ocs/index.php/ICWSM/ICWSM10/paper/viewFile/1536/1842>.

Ringsquandl, Martin, and Dušan Petković. "Anal yzing Political Sentiment on Twit ter."*University of Applied Sciences Rosenheim* (n.d.): n. pag. *ACL Anthology*. Web.

Wang, Yu, Tom Clark, Jeffrey Staton, and Eugene Agichtein. "Towards Tracking Political Senti ment through Microblog Data." *Emory Univer sity* (n.d.): n. pag. *ACL Anthology*. Web.

Wang, Hao, Dogan Can, Abe Kazemzadeh, Francois Bar, and Shrikanth Narayanan. "A System for Real-time Twitter Sentiment Analysis of 2012 U.S. Presidential Election Cycle." (n.d.): n. pag. ACLWeb Anthology. Web. <http://www.aclweb.org/anthology/P12-3020>.