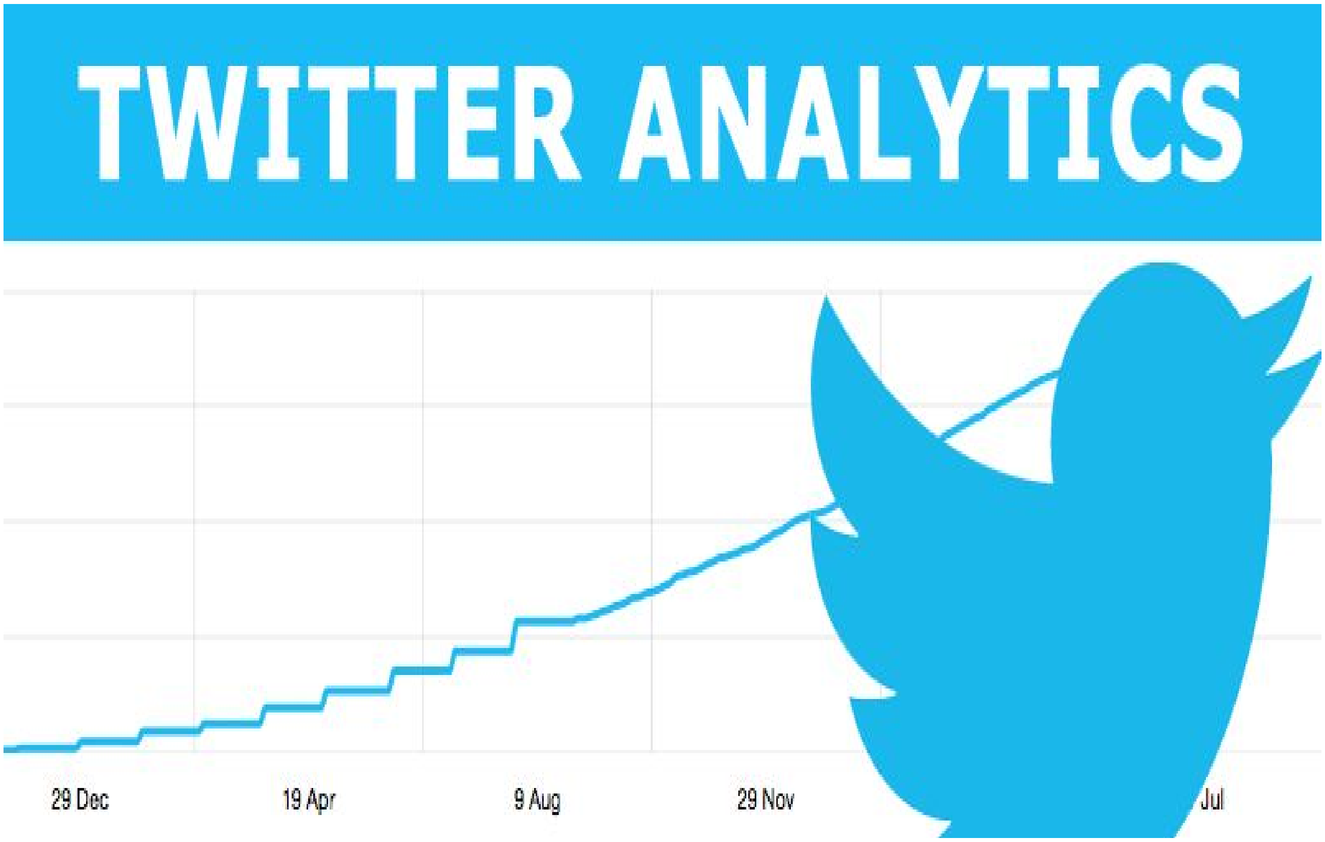


Twitter Sentiment Tracking:

Predictive Analytics in the Age of Social Media

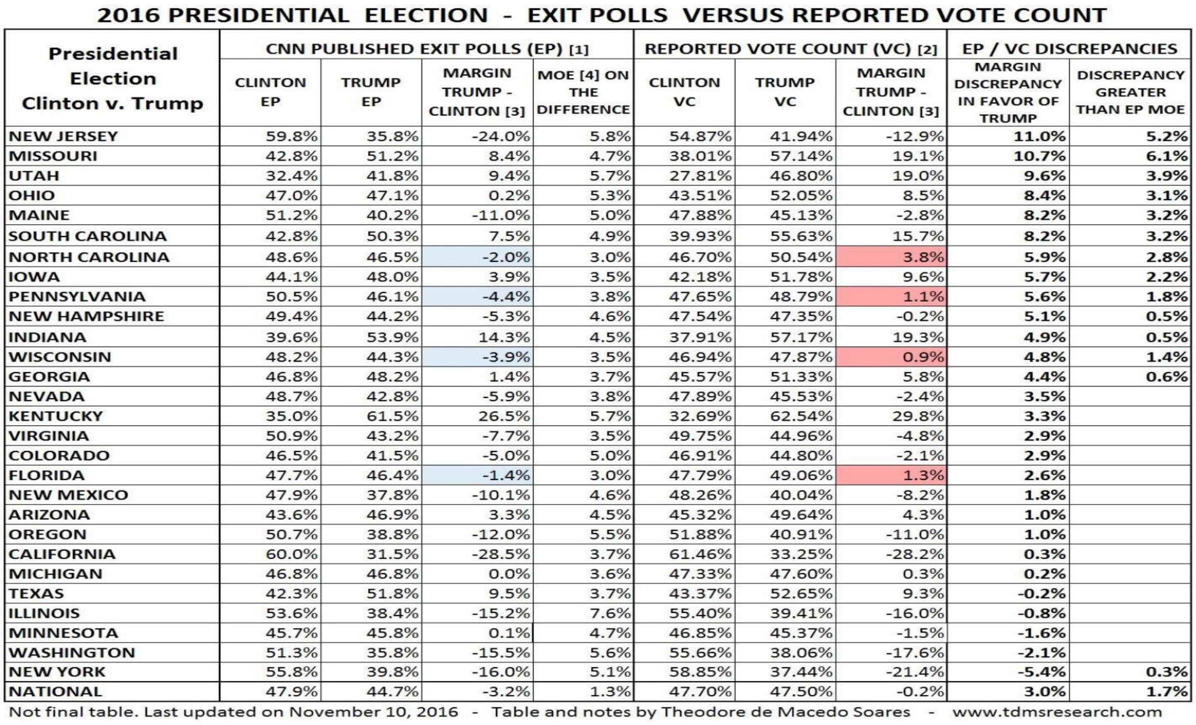
EN.600.466 Informational Retrieval and Web Agents

Jeong eun (Hailey) Lee, jlee562@jhu.edu

Arpan Ghosh, aghosh17@jhu.edu

Overview

The United States election of 2016 ushered in a new era of politics for the United States, but it also shocked the world, particularly for those in the field of predictive analytics. Almost every major polling site predicted between an 80% to over 99% landslide victory for the Democratic nominee Hillary Clinton (fig. 1), and others predicted a Clinton victory even if the margins weren't as high. Yet the reality on November 9th were electoral results that were very far from the predictions. Particularly, states that were unilaterally predicted to go blue for Hillary, that ultimately cost her the election, went to Trump in a decisive electoral victory (i.e. Pennsylvania, Wisconsin, and Michigan, and the wild-cards North Carolina and Florida). The predictive analytic business relies heavily on quantitative data, as qualitative data is hard enough to decipher. Major media outlets and the most trusted polling organizations suffered a significant embarrassment, and the question that went through the minds of everyone in the business of predictive and accurate data analytics were whether conventional polling mechanisms were still accurate, or was a landslide Trump victory truly a statistical anomaly?



2016 PRESIDENTIAL ELECTION - EXIT POLLS VERSUS REPORTED VOTE COUNT

Presidential Election Clinton v. Trump	CNN PUBLISHED EXIT POLLS (EP) [1]				REPORTED VOTE COUNT (VC) [2]			EP / VC DISCREPANCIES	
	CLINTON EP	TRUMP EP	MARGIN TRUMP - CLINTON [3]	MOE [4] ON THE DIFFERENCE	CLINTON VC	TRUMP VC	MARGIN TRUMP - CLINTON [3]	MARGIN DISCREPANCY IN FAVOR OF TRUMP	DISCREPANCY GREATER THAN EP MOE
NEW JERSEY	59.8%	35.8%	-24.0%	5.8%	54.87%	41.94%	-12.9%	**11.0%**	**5.2%**
MISSOURI	42.8%	51.2%	8.4%	4.7%	38.01%	57.14%	19.1%	**10.7%**	**6.1%**
UTAH	32.4%	41.8%	9.4%	5.7%	27.81%	46.80%	19.0%	**9.6%**	**3.9%**
OHIO	47.0%	47.1%	0.2%	5.3%	43.51%	52.05%	8.5%	**8.4%**	**3.1%**
MAINE	51.2%	40.2%	-11.0%	5.0%	47.88%	45.13%	-2.8%	**8.2%**	**3.2%**
SOUTH CAROLINA	42.8%	50.3%	7.5%	4.9%	39.93%	55.63%	15.7%	**8.2%**	**3.2%**
NORTH CAROLINA	48.6%	46.5%	-2.0%	3.0%	46.70%	50.54%	3.8%	**5.9%**	**2.8%**
IOWA	44.1%	48.0%	3.9%	3.5%	42.18%	51.78%	9.6%	**5.7%**	**2.2%**
PENNSYLVANIA	50.5%	46.1%	-4.4%	3.8%	47.65%	48.79%	1.1%	**5.6%**	**1.8%**
NEW HAMPSHIRE	49.4%	44.2%	-5.3%	4.6%	47.54%	47.35%	-0.2%	**5.1%**	**0.5%**
INDIANA	39.6%	53.9%	14.3%	4.5%	37.91%	57.17%	19.3%	**4.9%**	**0.5%**
WISCONSIN	48.2%	44.3%	-3.9%	3.5%	46.94%	47.87%	0.9%	**4.8%**	**1.4%**
GEORGIA	46.8%	48.2%	1.4%	3.7%	45.57%	51.33%	5.8%	**4.4%**	**0.6%**
NEVADA	48.7%	42.8%	-5.9%	3.8%	47.89%	45.53%	-2.4%	**3.5%**	
KENTUCKY	35.0%	61.5%	26.5%	5.7%	32.69%	62.54%	29.8%	**3.3%**	
VIRGINIA	50.9%	43.2%	-7.7%	3.5%	49.75%	44.96%	-4.8%	**2.9%**	
COLORADO	46.5%	41.5%	-5.0%	5.0%	46.91%	44.80%	-2.1%	**2.9%**	
FLORIDA	47.7%	46.4%	-1.4%	3.0%	47.79%	49.06%	1.3%	**2.6%**	
NEW MEXICO	47.9%	37.8%	-10.1%	4.6%	48.26%	40.04%	-8.2%	**1.8%**	
ARIZONA	43.6%	46.9%	3.3%	4.5%	45.32%	49.64%	4.3%	**1.0%**	
OREGON	50.7%	38.8%	-12.0%	5.5%	51.88%	40.91%	-11.0%	**1.0%**	
CALIFORNIA	60.0%	31.5%	-28.5%	3.7%	61.46%	33.25%	-28.2%	**0.3%**	
MICHIGAN	46.8%	46.8%	0.0%	3.6%	47.33%	47.60%	0.3%	**0.2%**	
TEXAS	42.3%	51.8%	9.5%	3.7%	43.37%	52.65%	9.3%	**-0.2%**	
ILLINOIS	53.6%	38.4%	-15.2%	7.6%	55.40%	39.41%	-16.0%	**-0.8%**	
MINNESOTA	45.7%	45.8%	0.1%	4.7%	46.85%	45.37%	-1.5%	**-1.6%**	
WASHINGTON	51.3%	35.8%	-15.5%	5.6%	55.66%	38.06%	-17.6%	**-2.1%**	
NEW YORK	55.8%	39.8%	-16.0%	5.1%	58.85%	37.44%	-21.4%	**-5.4%**	**0.3%**
NATIONAL	47.9%	44.7%	-3.2%	1.3%	47.70%	47.50%	-0.2%	**3.0%**	**1.7%**

Not final table. Last updated on November 10, 2016 - Table and notes by Theodore de Macedo Soares - www.tdmsresearch.com

Fig. 1 Poll vs Election Results Discrepancies

Procedure

We wanted to find out the answer to the above question. To focus on a specific testable simulation, we focused on the following: **Could conducting a sentiment analysis of tweets have predicted an electoral victory for Donald Trump?** Moreover, the beauty behind analyzing language to track sentiment is that we ultimately end up with quantitative data using a qualitative approach. To do this, we relied on the following broad categorizations for modeling the English language to categorize tweets as being **pro-Trump/anti-Clinton or pro-Clinton/anti-Trump.** By scraping a random 100,000 tweets in the US for the month of July, October, and during election week (November 1st- November 8th), we will have collected a total of 600,000 tweets (300,000 with the keyword Trump and 300,000 with Clinton). We will analyze our raw results in the following manner: analyze the number of tweets that fall in pro-Trump and pro-Clinton camps by weighting the hashtags, region weighting, and term weighting, as well as run a geo-tag location extraction on the JSON of each tweet. The goal is to map the tweets, with heavy focus on the five shocking states where mainstream media outlets were confident for a Clinton victory yet went Trump: Pennsylvania, Wisconsin, Michigan, North Carolina. The extension of our focus question to the data is: **Could analyzing tweets have predicted an electoral victory for Trump in the rust belt and swing states?** Our prediction is: **Yes, twitter sentiment analysis is a powerful supplement towards accurately prediction of electoral results, and would have predicted a Trump electoral victory.**

Goals

1. Scrape past tweets by circumventing Twitter's API "last 7-days only" limitation
2. Term weighting, hashtag weighting, ambiguous weighting, reg-exp, word/context disambiguation
3. Geotagging/location extraction from tweets in JSON format
4. Data analysis, modeling, conclusion

Specifications

APIs, Twitter Scraping, Crawling & Accurate Dataset

Empirical Evaluation (Term weighting, hashtag weighting, pos/neg word disambiguation)

Problems/Concerns

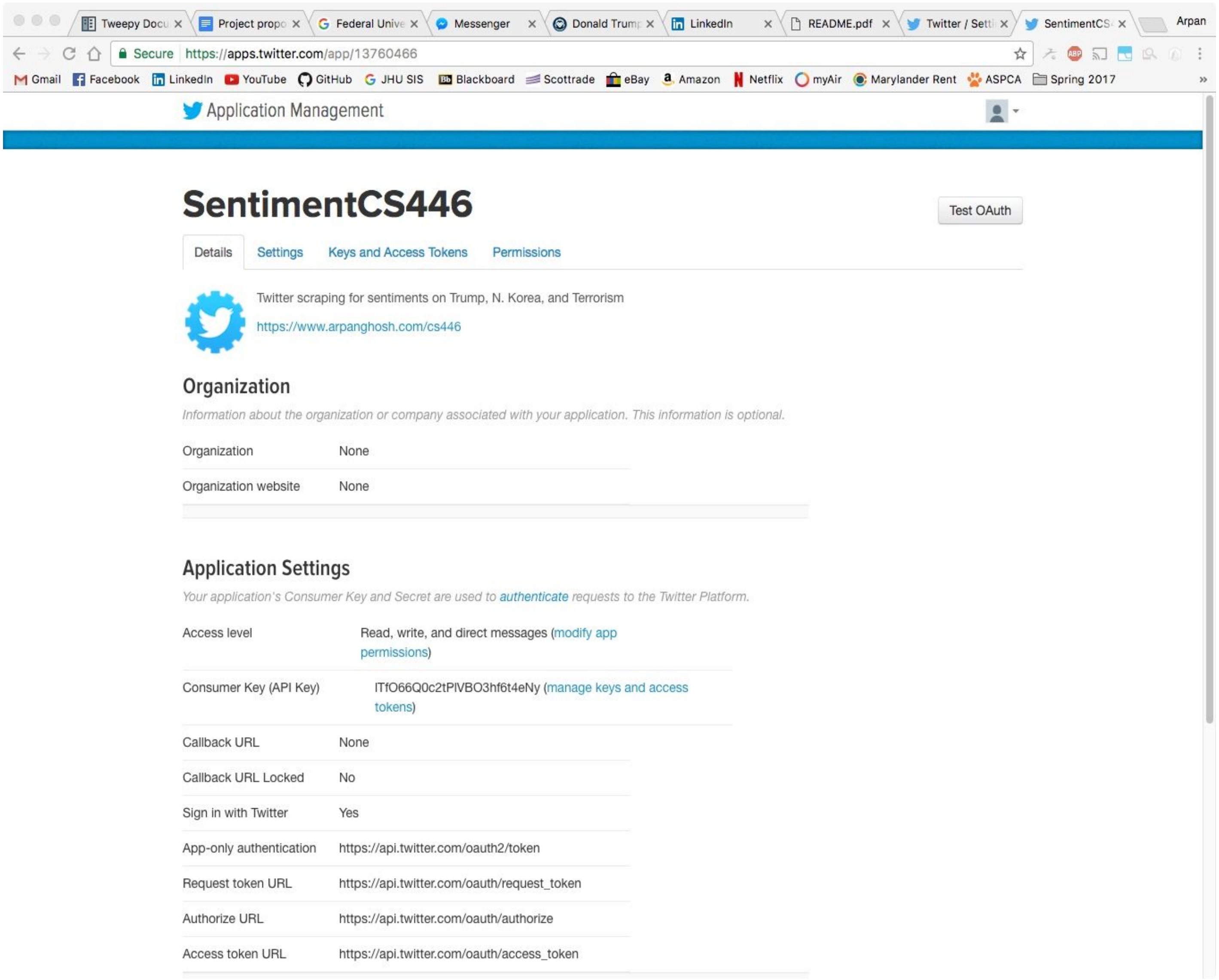
Conclusion (Was our hypothesis correct?)

Milestones

I. Successful Twitter scraping

We had to find a way to retrieve consistent data for accurate analytical purposes. If our data is skewed, then no matter how well our weighting functions are written, our results will not make sense nor be accurate. At the same time, we were aware of our limitations. Using Twitter's API solely would limit our tweet scraping to the last 7 days. Clearly this wouldn't be an issue had we been scraping actively during the election cycle, but we needed to find a way to scrape beyond the last 7 days, for whatever date we decided to input. We did this by heavily modifying the existing API that bypasses the limitations of Twitter's official API, of a computer scientist at the Federal University of Piaui in Brazil, using a tool called, "GetOldTweets-Python," which extracts the raw JSON requests from any tweet, passing in a keyword, user ID, hashtags, mentions, tweets ID, etc: (link: https://github.com/Jefferson-Henrique/GetOldTweets-python).

Then we had to authorize our Twitter account and register it as an organization, and providing the correct authentication tokens to permit our libraries and code to work seamlessly with Twitter's API. This particular authentication required a lot of work and code to make sure the SSL security was being achieved as per Twitter's rules, as well as all our applications were properly authenticated to reach the Twitter API as per our specific organization:



SentimentCS446

Twitter scraping for sentiments on Trump, N. Korea, and Terrorism

https://www.arpanghosh.com/cs446

Organization

Information about the organization or company associated with your application. This information is optional.

Organization	None
Organization website	None

Application Settings

Your application's Consumer Key and Secret are used to authenticate requests to the Twitter Platform.

Access level	Read, write, and direct messages (modify app permissions)
Consumer Key (API Key)	ITfO66Q0c2tPlVBO3hf6t4eNy (manage keys and access tokens)
Callback URL	None
Callback URL Locked	No
Sign in with Twitter	Yes
App-only authentication	https://api.twitter.com/oauth2/token
Request token URL	https://api.twitter.com/oauth/request_token
Authorize URL	https://api.twitter.com/oauth/authorize
Access token URL	https://api.twitter.com/oauth/access_token

Although this solution seemed it would have worked perfectly out of the bag, it did not. We had to dramatically modify the criteria to work with Twitter's native API update that was set in place earlier this year. This meant we had to find a way to return the list of tweets retrieved by query search, bound dates, and looping to find a random until reaching 100,000 tweets. We ran the following commands to retrieve 100,000 tweets from each of the following: July 2016 keyword: Trump, July 2016 keyword Clinton:, October 2016 keyword: Trump, October 2016 keyword: Clinton, Election Week keyword: Trump, Election week keyword: Clinton.

```
python Trump_July.py --querysearch "trump" --since 2016-07-01 --until 2016-07-31 --maxtweets 100000
python Trump_Oct.py --querysearch "trump" --since 2016-10-01 --until 2016-10-31 --maxtweets 100000
python Trump_Nov.py --querysearch "trump" --since 2016-11-01 --until 2016-11-09  --maxtweets 100000
python Clinton_July.py --querysearch "trump" --since 2016-07-01 --until 2016-07-31 --maxtweets 100000
python Clinton_Oct.py --querysearch "trump" --since 2016-10-01 --until 2016-10-31 --maxtweets 100000
python Clinton_Nov.py --querysearch "trump" --since 2016-11-01 --until 2016-11-09  --maxtweets 100000
```

Fig. I Please look at the last section of this document to view the code (code snippet #1)

The other problems we realized were the following: we have limited computation power. If we were able to scrape literally *every single* tweet, as in all ~300 million+ associated with the election, then we could have scraped each tweet by hashtag alone, and keep a counter on the number of pro-Trump and pro-Clinton hashtags. This, however, would have been impossible with our computers. Moreover, had we even been successful in the extraction of every tweet, running the weighting similarities would have been unreasonable. So we focused on just finding 100,000 random tweets in each category, for a total of 600,000 tweets and running our weighting functions on it.

CODE

snippet of keyword tweet, find JSON, extract relevant information

```
json = TweetManager.getJsonReponse(tweetCriteria, refreshCursor, cookieJar)
                    if len(json['items_html'].strip()) == 0:
                          break

                    refreshCursor = json['min_position']
                    tweets = PyQuery(json['items_html'])('div.js-stream-tweet')
```

Geolocation Extraction

We were able to extract comma separated values for the associated tweet text with the tweet ID and the user ID. Afterwards, what we did was remove tweets from the same user ID (as this could skew data, although the effect it would have on the data would be very minimal). Afterwards, we copy the tweet IDs from a single column in Microsoft Excel and put it in a new spreadsheet. We take that spreadsheet and run it through a curl script that we built specifically for the following that are special for Twitter JSONs: geo-location, full_name, place, JSON extractions, a module called Twurl, an OAuth-enabled curl for the Twitter API (https://github.com/twitter/twurl).

Running a bash script, of the following nature prints out the three parameters that we want for each tweet, and here is the python script (here is a short snippet, the full code is in the attached submission and end of this pdf):

… **twurl_requests.sh > results.txt:**

```
twurl “/1.1/statuses/show/792140163514327000.json”
printf $’\n’
printf $’\n’
twurl “/1.1/statuses/show/792140163514327041.json”
```

…

... **location_extraction.py**

```
for tweet in results:
     print (tweet.get(‘id’,{}).get(‘location’,{}))
    print (tweet.get(‘id’,{}).get(‘full_name’,{}))
    print (tweet.get(‘id’,{}).get(‘place’,{}))
```

After we find the location, we save it to a CSV file and append it to their corresponding tweet IDs on a full CSV that has the following data: tweet ID, tweet text, tweet location, tweet place.

Then, after running the text data using our empirical evaluation programs (detailed in the next section), we assign each tweet with T or C, to indicate pro Trump (T) or pro Clinton (C) . Based on the T or C classification per location, we map the data, specifically parsing for the following states (or cities in the state): Michigan, Wisconsin, Pennsylvania, North Carolina (and txt files of cities in each state for the matching).

[INSERT DATA GRAPH HERE]

II. Empirical Evaluation (Term weighting, hashtag weighting, pos/neg word disambiguation)

Moreover, we had to write a list of hashtags, and positive and negative words, while being mindful of the language people use on Twitter. Also, we had to figure out what to do with ambiguous, or neutral, tweets. Afterwards, we put them through a few functions: hashtag_weighting() and term_weighting(). The function, hashtag_weighting() works by assigning a +5 or -5 for obvious hashtags that are pro-Trump or anti-Trump and pro-Clinton and anti-Clinton.

- A short example from our full lexicon of pro-Trump/anti-Clinton tweets are the following: #GoHillary, #ImWithHer, #DumpTrump, #HillaryForPresident, #amerikkka, #PutinsPuppet, etc.
- A short example from our full lexicon of pro-Clinton/anti-Trump tweets are the following: #MAGA, #MakeAmericaGreatAgain, #VoteTrump, #NeverHillary, #HillaryForPrison, #LockHerUp

In the lexicons for each, we are 100% sure that they indicate either pro-Trump or pro-Clinton. This leads us to examining ambiguous hashtags, like #Trump or #Hillary, because users could be referencing a candidate but not necessarily endorsing either in the tweet. For situations like this we conduct a region weighting. Similarly, when looking through keywords of just "Trump" and "Clinton" (that aren't in hashtags but clearly refer to the election), we perform term weighting by looking at +5 and -5 words around the terms and trying to correlate positive/negative values with number of appearances of each. The approach we take is similar to what we learned in class for homework #2.

Here is an example of a pro-Trump tweet:

"I follow Mr. Trump at all of his rallies by watching them...He is a lion-hearted warrior who inspires hope."

Here is an example of a pro-Clinton tweet:

"Trump's lies about #hillarysemail are so ridiculous and so dishonest that if you believe them, maybe you're too stupid to vote! #VoteBlue

Here is an example of an ambiguous tweet:

"And now Harry Reid has said the FBI is withholding evidence Trump and Manafort worked with Russia. All very 'interesting.' 9/9"

In the situations above, our scraper easily grabbed these tweets just by the keyword Trump or Hillary. In situations where the words Putin and Russia were referenced to a Trump/Clinton keyword tweet, we had to add a few more words to the full list of negative words, specifically to deal with this period of the election:

- Here are some words that have to do with Putin/Russia but would need further refinement to check whether anti-Trump or Clinton: colluded, hack, involve, cheated, war, interfere, WTF, blame, thanks, truth, ties, investigate, puppet, coverup, aided, aided and abetted, abetted, friend, traitor, criminal, end of democracy

And then of course, we had an entire corpus of words that correlate to positive and negative sentiment (NEGATIVE opinion words (or sentiment words).). Here is a short snippet from those text files (the actual files range from words A-Z, but a snippet of positive and negative here just displays a few choice words to provide an example).

GitHub links:

https://gist.github.com/mkulakowski2/4289441, and https://gist.github.com/mkulakowski2/4289437

Positive	Negative
... assassin assassinate assault assult astray asunder atrocious atrocities atrocity atrophy attack attacks audacious audaciously audaciousness audacity audiciously austere authoritarian autocrat ...	… homage honest honesty honor honorable honored honoring hooray hopeful hospitable hot hotcake hotcakes hottest hug humane humble humility humor humorous ...

III. Problems, Concerns, Workarounds, etc.

Some legitimate problems we encountered had to do with something quite significant: not every tweet had a location set by the user NOR location-tracking enabled on the phone through which the tweet may have been sent. This meant that not all of our 600,000 tweets were marked on the map. Moreover, out of these 600,000 tweets, only a small fraction were sent from the swing states in section. In no way can we conclude that our results are 100% accurate for the entire group of users who have tweeted from those swing states. But we were able to find a consistent group of Twitter users from the four swing states (Michigan, Florida, Pennsylvania, Wisconsin) through each data set (July, October, and Election Week), and we were able to conclude something useful. However, what matters the most is that our program could handle as many tweets as needed, so if we had the computational power to run and extract data from hundreds of million tweets in JSON format, our information retrieval mechanisms and sentiment analysis/geo-location mapping of the tweets would be accurate.

To aid in this process, we were also fortunate to find a dataset of tweets that's had the keyword “Trump” and “Clinton,” along with the location name set by the user. The problems of this were obvious: many users chose to include their locations, which we will take as being true, but many also stated names of absurd locations perhaps to be humorous or mask their true location, such as “Location: Middle of Nowhere”. In situations like this, we used the tweet to track sentiment since we know the tweets were all scraped from the United States, but we ignored this tweet in the location modeling.

Here is how we crawled tweets based on location:

Using jq, a lightweight and flexible command-line JSON processor, we were able to combine the power of twurl with the necessity to find tweets based on location, a keyword, and language. We created a bash script to feed the following format, changing the location to match the latitude and longitude that can be parsed in the proper Twitter API format for location, in the following locations (swing states and states that had a strong prediction for Clinton but went

Trump), Ohio, North Carolina, Florida, Pennsylvania, Michigan, Wisconsin (set to varying radius depending on the size of the state from the most middle point city in the states). For example, 40°25'02.3"N 82°54'25.6"W in Ohio would be the coordinates for the middle point city (Ashley, Ohio), set to a 200 mile radius, of the tweet denoted below in the snippet. If the tweet location falls in the radius of Ohio, then return true and tweet ID of the tweet:

CODE Snippet (Location Extraction)

```
twurl -t -d bool=true place.name=‘ohio’ locations=40.25,82.54,2.3,25.56
locations.radius=200m language=en id=792140163514327000 -H
stream.twitter.com/1.1/statuses/filter.json | jq '.text' > ohio.txt
```

IV. Data Modeling and Analysis

Data Modeling

Try 1:

For this model, I used stemming, positive/negative dictionaries, term weighting, and hashtag detection, without any training model.

- With the train data set, the positive tweets are 606,296 and negative tweets are 1,156,142.
- I got: the positive tweets are 488,676 and negative tweets are 684,486.

This model had low recall rate, but decent accuracy.

Try 2:

Change the positive/negative dictionaries to the train dataset and store into train_dict for term weighting. For example, if “I hate Trump” negative tweet with value -0.6, then “hate” “Trump” will be in train_dict for -0.6 weighting.

- For 100 datasets, the ground truth was (positive 23 negative 57). With this model, out of 100 train datasets, I got (positive 16 and negative 55)
- For 300 datasets, the ground truth was (positive 69 negative 139) With this model, out of 600 train datasets, I got (positive 20 and negative 181)
- With the train data set, the positive tweets are 606296 and negative tweets are 1156142. With my model, I got (positive 125087 negative 1488857)

This model had closer recall and accuracy for smaller sets, but for larger sets, recall and precision went down.

- With this model, when I ran the Trump_july_2016_tweets_only.csv, I get 1461 pro-trump and 14593 anti-trump
- Also, when I ran the Trump_nov_2016_tweets_only.csv, I get 7371 pro-trump and 744663 anti-trump.

Try 3:

I realized Trump and Clinton or Hillary appear a lot and also in the train_dict, which could be a deciding factor. Therefore, I decided to remove these keywords this time.

- For 300 datasets, the ground truth was 69 positive and 139 negative tweets. With this model, out of 600 train datasets, the positive tweets are 62 and negative tweets are 139.
- For 600 datasets, the ground truth was (positive 116, negative 300). With this model, I got (positive 115, negative 280)
- For entire datasets, the ground truth is (positive 606296, negative 1156142). With this model, I got (positive 738098, negative 882897) (if no limit on positive/negative train datasets, then positive 175627, negative 1445567)
- With this model, when I ran Trump_nov_2016_tweets_only, I got (positive 40582, negative 37753, total 99943)
- With this model, when I ran Trump_july_2016_tweets_only, I got (positive 7104, negative 8821, total 18238)
- With this model, when I ran Clinton_nov_2016_tweets_only, I got (positive 20216, negative 20123, total 99926)
- With this model, when I ran Clinton_july_2016_tweets_only, I got (positive 20287, negative 29312, total 99801)
- With this model, when I ran Swing_states_Trump, I got (positive 478, negative 680, total 1597)
- With this model, when I ran Swing_states_Clinton, I got (positive 170, negative 156, total 1903)

Try 4

With couple changes in language model (add pro and anti of other candidates into account)

- For 300 datasets, the ground truth was 68 positive and 138 negative tweets. With this model, I got (positive 70, negative 134)
- For 600 datasets, the ground truth was (positive 116, negative 300). With this model, I got (positive 129, negative 280)

- For entire datasets, the ground truth is (positive 606296, negative 1156142). With this model, I got (positive 670474, negative 1003581) (if no limit on positive/negative train datasets, then positive 175627, negative 1445567)
- With this model, when I ran Swing_states_Trump, I got (positive 480, negative 691, total 1597)
- With this model, when I ran Trump_nov_2016_tweets_only, I got (positive 40582, negative 37753, total 99943)
- With this model, when I ran Trump_july_2016_tweets_only, I got (positive 7094, negative 9001, total 18238)

By far, the highest recall/precision rate for Trump model.

Try 5

Change of hillary model: used hand-labelled data to train model, and extract lines with high-confidence level to keep adding onto train data.

- From 100 hand-labelled datasets, the label has 18 positive and 35 negative tweets. When I trained the model, I ended up getting 24 positive and 41 negative tweets.
- With this model, when I ran Clinton_july_2016_tweets_only, I got (positive 27794 and negative 45001, total 99801)
- With this model, when I ran Clinton_nov_2016_tweets_only, I got (positive 3365 and negative 77244, total 99926)
- With this model, when I ran Swing_states_Clinton, I got (positive 98 and negative 1042, total 1903)

Try 6

Change of hillary model: allow the same number of negative and positive tweets with confidence level to be written into the train dataset

- From 100 hand-labelled datasets, the label has 18 positive and 35 negative tweets. When I trained the model, I ended up getting 24 positive and 41 negative tweets.
- With this model, when I ran Clinton_july_2016_tweets_only, I got (positive 29353 and negative 45001, total 99801)
- With this model, when I ran Clinton_nov_2016_tweets_only, I got (positive 24213 and negative 52493, total 99926)
- With this model, when I ran Swing_states_Clinton, I got (positive 349 and negative 792, total 1903)

By far, the highest recall/precision rate for Clinton model.

Analysis

Instead of using positive and negative dictionary, we decided to train our model based on the labeled data. Since tweets normally consist of few words (besides stop words and hashtags) and political tweets tend to have lots of sarcasm, negative/positive words in the tweets do not necessarily define the sentiments of tweets properly.

Luckily, we were able to get 300 MB labeled data for tweets with keyword "Trump". With this train data, we have created a dictionary of all the words frequently mentioned related to Trump based with weighting of cumulation of values from train data. For example, if a tweet from Train data, *"I follow Mr. Trump at all of his rallies by watching them...He is a lion-hearted warrior who inspires hope."* has a value of 0.7, then all the words in this tweet that are not common words (e.g. {follow, rallies, watching, lion-hearted, warrior, inspire, hope}) have equal values of 0.7. If a word "follow" appears as positive connotation for multiple tweets with values (0.7, 0.4, 0.5 …) then, eventual weight for word "follow" will sum all the values and be weighted more toward positive than negative.

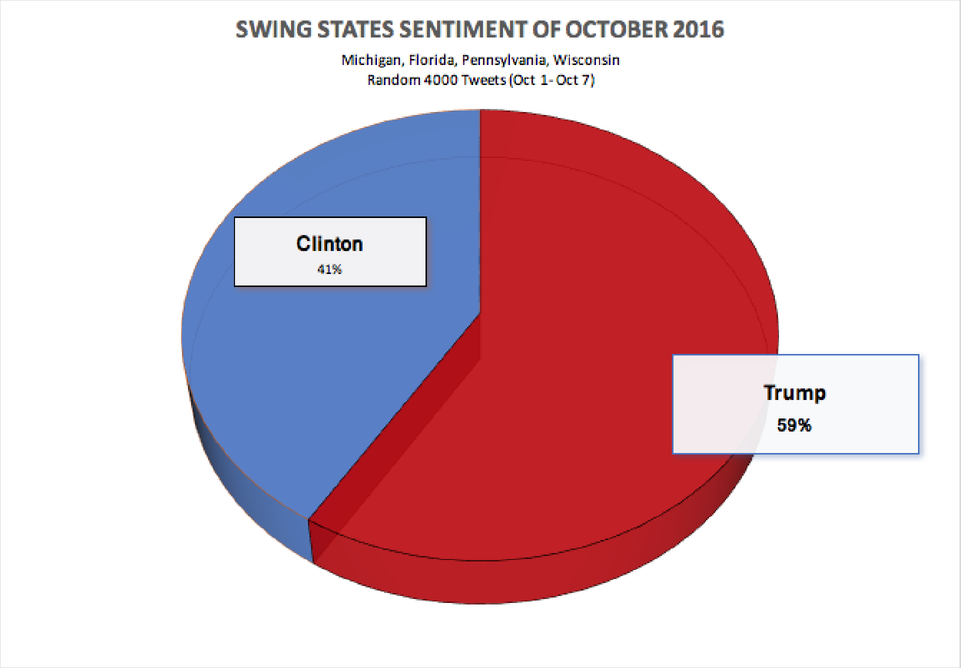
On the other hand, we were unable to find labeled dataset for Clinton. Thus, to improve the accuracy, we decided to do semi-supervised learning. After labeling first 100 data (called clinton_train_original.csv), we ran random dataset and labeled them to become train dataset, and by running couple times, we were able to get decent number of training data. However, in this case, it is hard to control the accuracy (or precision/recall model), since after first 100 dataset, we lack ground truth to find the accuracy of model.

Tweets, especially politically-inclined tweets, are hard to analyze their sentiments with thesaurus, since the words appear in tweets are more colloquial than those that appear in regular documents. Therefore, with train data, we were able to detect more sentiments and increase accuracy. Here are some tables and graphs demonstrating our findings:

	Trump_July	Trump_Nov	Clinton_July	Clinton_Nov
Pro	7094	40582	29353	24213
Anti	9001	37753	45001	52493
Total	18238	99943	99801	99926

V. Swing States

Batch #1



Batch #2

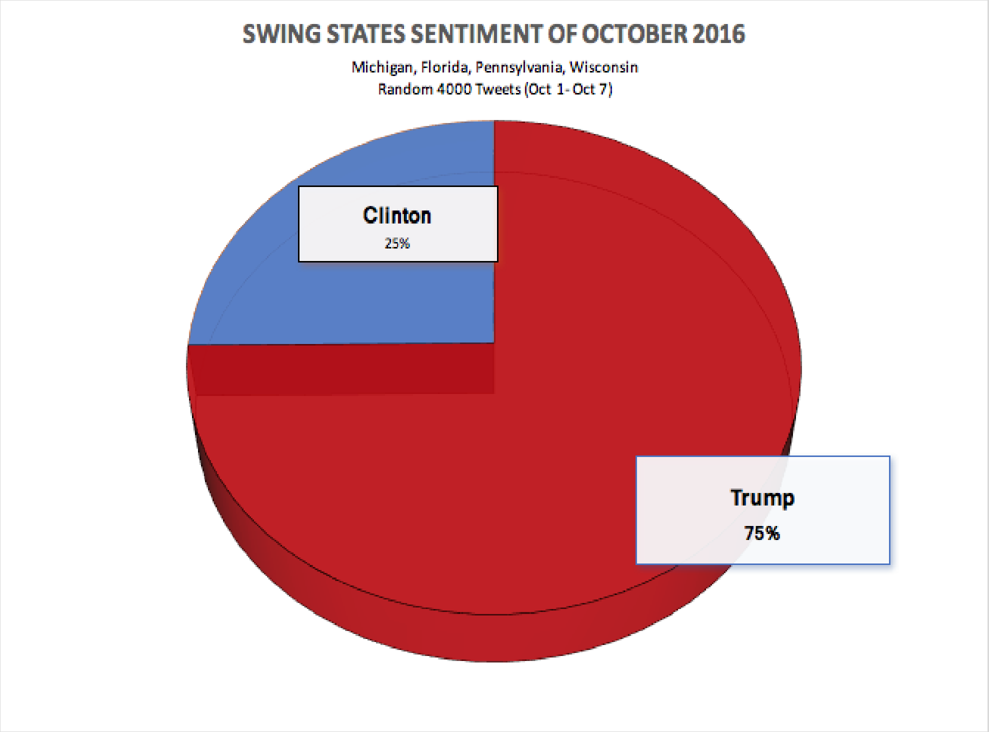


Table: Batch #2

	#Trump	**#Clinton**
Pro	179	128
Anti	688	1006
Ambiguous	429	774
Detected Set	ALL	ALL
Entire Set	1596	1903

VI. Conclusion

Our results show that in the swing states (Michigan, Florida, Pennsylvania, and Wisconsin), our twitter detection model found widely more pro Trump tweets than Clinton. We ran this on two random batches of 4000 tweets each for a total of 8000 and we found both times that more people ended up being pro-Trump than pro-Clinton. **Batch #2 results were not even remotely similar to how the actual election went, but Batch #1 was shockingly closer to the accuracy of the election result.** That being said, our dataset was only 4000 tweets and the detection of these tweets did not take into account the cities of the swing states, nor the counties. This means there could have been an uneven number of tweets coming from democratic/more liberal areas in the swing states vs. the more conservative. *Unfortunately, Twitter does not have accurate tracking of past user data down precisely by a county, town, or municipality data.*

CODE

TweetManager (JSON Extraction)

```
import urllib,urllib2,json,re,datetime,sys,cookielib
from .. import models
from pyquery import PyQuery

class TweetManager:

	def __init__(self):
		pass

	@staticmethod
	def getTweets(tweetCriteria, receiveBuffer = None, bufferLength = 100):
		refreshCursor = ''

		results = []
		resultsAux = []
		cookieJar = cookielib.CookieJar()

		if hasattr(tweetCriteria, 'username') and
(tweetCriteria.username.startswith("\'") or
tweetCriteria.username.startswith("\"")) and (tweetCriteria.username.endswith("\'")
or tweetCriteria.username.endswith("\"")):
			tweetCriteria.username = tweetCriteria.username[1:-1]

		active = True

		while active:
			json = TweetManager.getJsonReponse(tweetCriteria,
refreshCursor, cookieJar)
			if len(json['items_html'].strip()) == 0:
				break

			refreshCursor = json['min_position']
			tweets = PyQuery(json['items_html'])('div.js-stream-tweet')

			if len(tweets) == 0:
				break

			for tweetHTML in tweets:
				tweetPQ = PyQuery(tweetHTML)
				tweet = models.Tweet()

				usernameTweet =
tweetPQ("span.username.js-action-profile-name b").text();
				txt = re.sub(r"\s+", " ",
tweetPQ("p.js-tweet-text").text().replace('# ', '#').replace('@ ', '@'));
```

```
                retweets = int(tweetPQ("span.ProfileTweet-action--retweet
span.ProfileTweet-actionCount").attr("data-tweet-stat-count").replace(",", ""));
                favorites =
int(tweetPQ("span.ProfileTweet-action--favorite
span.ProfileTweet-actionCount").attr("data-tweet-stat-count").replace(",", ""));
                dateSec = int(tweetPQ("small.time
span.js-short-timestamp").attr("data-time"));
                id = tweetPQ.attr("data-tweet-id");
                permalink = tweetPQ.attr("data-permalink-path");

                geo = ''
                geoSpan = tweetPQ('span.Tweet-geo')
                if len(geoSpan) > 0:
                    geo = geoSpan.attr('title')

                tweet.id = id
                tweet.permalink = 'https://twitter.com' + permalink
                tweet.username = usernameTweet
                tweet.text = txt
                tweet.date = datetime.datetime.fromtimestamp(dateSec)
                tweet.retweets = retweets
                tweet.favorites = favorites
                tweet.mentions = "
".join(re.compile('(@\\w*)').findall(tweet.text))
                tweet.hashtags = "
".join(re.compile('(#\\w*)').findall(tweet.text))
                tweet.geo = geo
                tweet.author_id = user_id
          tweet.tweetPQ = tweetPQ
          tweet.rawhtml = tweetHTML
          tweet.tweets = tweets
          tweet.alljson = json

                results.append(tweet)
                resultsAux.append(tweet)

                if receiveBuffer and len(resultsAux) >= bufferLength:
                    receiveBuffer(resultsAux)
                    resultsAux = []

                if tweetCriteria.maxTweets > 0 and len(results) >=
tweetCriteria.maxTweets:
                    active = False
                    break


        if receiveBuffer and len(resultsAux) > 0:
            receiveBuffer(resultsAux)

        return results

    @staticmethod
```

```
      def getJsonReponse(tweetCriteria, refreshCursor, cookieJar):
            url =
"https://twitter.com/i/search/timeline?f=tweets&q=%s&src=typd&max_position=%s"

            urlGetData = ''
            if hasattr(tweetCriteria, 'username'):
                  urlGetData += ' from:' + tweetCriteria.username

            if hasattr(tweetCriteria, 'since'):
                  urlGetData += ' since:' + tweetCriteria.since

            if hasattr(tweetCriteria, 'until'):
                  urlGetData += ' until:' + tweetCriteria.until

            if hasattr(tweetCriteria, 'querySearch'):
                  urlGetData += ' ' + tweetCriteria.querySearch

            if hasattr(tweetCriteria, 'topTweets'):
                  if tweetCriteria.topTweets:
                        url =
"https://twitter.com/i/search/timeline?q=%s&src=typd&max_position=%s"

            url = url % (urllib.quote(urlGetData), refreshCursor)

            headers = [
                  ('Host', "twitter.com"),
                  ('User-Agent', "Mozilla/5.0 (Windows NT 6.1; Win64; x64)"),
                  ('Accept', "application/json, text/javascript, */*; q=0.01"),
                  ('Accept-Language', "de,en-US;q=0.7,en;q=0.3"),
                  ('X-Requested-With', "XMLHttpRequest"),
                  ('Referer', url),
                  ('Connection', "keep-alive")
            ]

            opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cookieJar))
            opener.addheaders = headers

            try:
                  response = opener.open(url)
                  jsonResponse = response.read()
            except:
                  print "Twitter weird response. Try to see on browser:
https://twitter.com/search?q=%s&src=typd" % urllib.quote(urlGetData)
                  sys.exit()
                  return

            dataJson = json.loads(jsonResponse)

            return dataJson

      @staticmethod
    def findLocation(username):
        url = "https://twitter.com/%s" % username

        headers = [
            ('Host', "twitter.com"),
            ('User-Agent', "Mozilla/5.0 (Windows NT 6.1; Win64; x64)"),
            ('Accept', "application/json, text/javascript, */*; q=0.01"),
            ('Accept-Language', "de,en-US;q=0.7,en;q=0.3"),
            ('X-Requested-With', "XMLHttpRequest"),
            ('Referer', url),
            ('Connection', "keep-alive")
        ]

        opener = urllib.request.build_opener(
            urllib.request.HTTPCookieProcessor(http.cookiejar.CookieJar()))
        opener.addheaders = headers

        try:
            response = opener.open(url)
            jsonResponse = response.read()
        except:
            print("Twitter weird response. Try to see on browser:
https://twitter.com/%s" % username)
            print("Unexpected error:", sys.exc_info()[0])
            sys.exit()
            return

        "span.ProfileHeaderCard-locationText"

        info = PyQuery(jsonResponse)

        geo_location = info("span.ProfileHeaderCard-locationText").text()

        return geo_location
```

Trump/Clinton__MONTH.py

```
# -*- coding: utf-8 -*-

import sys,getopt,got,datetime,codecs

def main(argv):

       if len(argv) == 0:
             print ('You must pass some parameters. Use \"-h\" to help.')
             return

       if len(argv) == 1 and argv[0] == '-h':
             print ("""\nTo use this jar, you can pass the folowing attributes:
    username: Username of a specific twitter account (without @)
       since: The lower bound date (yyyy-mm-aa)
       until: The upper bound date (yyyy-mm-aa)
 querysearch: A query text to be matched
   maxtweets: The maximum number of tweets to retrieve

 \nExamples:
 # Example 1 - Get tweets by username [barackobama]
 python Exporter.py --username "barackobama" --maxtweets 1\n

 # Example 2 - Get tweets by query search [europe refugees]
 python Exporter.py --querysearch "europe refugees" --maxtweets 1\n

 # Example 3 - Get tweets by username and bound dates [barackobama, '2015-09-10',
'2015-09-12']
 python Exporter.py --username "barackobama" --since 2015-09-10 --until 2015-09-12
--maxtweets 1\n

 # Example 4 - Get the last 10 top tweets by username
 python Exporter.py --username "barackobama" --maxtweets 10 --toptweets\n""")
             return

      try:
             opts, args = getopt.getopt(argv, "", ("username=", "since=", "until=",
"querysearch=", "toptweets", "maxtweets="))

             tweetCriteria = got.manager.TweetCriteria()

             for opt,arg in opts:
                    if opt == '--username':
                           tweetCriteria.username = arg

                    elif opt == '--since':
                           tweetCriteria.since = arg

                    elif opt == '--until':
                           tweetCriteria.until = arg

                    elif opt == '--querysearch':
                           tweetCriteria.querySearch = arg

                    elif opt == '--toptweets':
                           tweetCriteria.topTweets = True

                    elif opt == '--maxtweets':
                           tweetCriteria.maxTweets = int(arg)


             outputFile = codecs.open("output_got.csv", "w+", "utf-8")
```

```
outputFile.write('username;date;retweets;favorites;text;geo;mentions;hashtags;id;pe
rmalink')

                print ('Searching...\n')

                def receiveBuffer(tweets):
                        for t in tweets:
                                outputFile.write(('\n%s;%s;%d;%d;"%s";%s;%s;%s;"%s";%s' %
(t.username, t.date.strftime("%Y-%m-%d %H:%M"), t.retweets, t.favorites, t.text,
t.geo, t.mentions, t.hashtags, t.id, t.permalink)))
                        outputFile.flush();
                        print ('More %d saved on file...\n' % len(tweets))

                got.manager.TweetManager.getTweets(tweetCriteria, receiveBuffer)

        except arg:
                print ('Arguments parser error, try -h' + arg)
        finally:
                outputFile.close()
                print ('Done. Output file generated "output_got.csv".')

if __name__ == '__main__':
        main(sys.argv[1:])
```

TwitterClient.py (Sentiment tracking tool for checking purposes)

```
import re
import tweepy
from tweepy import OAuthHandler
from textblob import TextBlob

class TwitterClient(object):
    '''
    Generic Twitter Class for sentiment analysis.
    '''
    def __init__(self):
        '''
        Class constructor or initialization method.
        '''
        # keys and tokens from the Twitter Dev Console
        consumer_key = 'lGiR5932nSrahZijJXYDHIGln'
        consumer_secret = 'wY0qlSW8XNZXTv2IiOn6UOlVBKxAeZjJfMNy5913cXBlij3PBw'
        access_token = '859889729344614401-VcPKEMK2QwSsQQPE4mE9PjqJI1VuvOX'
        access_token_secret = 'dIgLfI9PLnqGn1cMvbXf0bpOrW3lYGjVrIQKETFxYUGck'

        # attempt authentication
        try:
            # create OAuthHandler object
            self.auth = OAuthHandler(consumer_key, consumer_secret)
            # set access token and secret
            self.auth.set_access_token(access_token, access_token_secret)
            # create tweepy API object to fetch tweets
            self.api = tweepy.API(self.auth)
        except:
            print("Error: Authentication Failed").encode('utf-8')

    def clean_tweet(self, tweet):
        '''
        Utility function to clean tweet text by removing links, special characters
        using simple regex statements.
        '''
        return ' '.join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)", "
", tweet).split())

    def get_tweet_sentiment(self, tweet):
        '''
        Utility function to classify sentiment of passed tweet
        using textblob's sentiment method
        '''
        # create TextBlob object of passed tweet text
        analysis = TextBlob(self.clean_tweet(tweet))
        # set sentiment
        if analysis.sentiment.polarity > 0:
            return 'positive'
        elif analysis.sentiment.polarity == 0:
            return 'neutral'
        else:
            return 'negative'

    def get_tweets(self, query, count = 10):
        '''
        Main function to fetch tweets and parse them.
        '''
        # empty list to store parsed tweets
        tweets = []

        try:
            # call twitter api to fetch tweets
            fetched_tweets = self.api.search(q = query, count = count)

            # parsing tweets one by one
            for tweet in fetched_tweets:
                # empty dictionary to store required params of a tweet
                parsed_tweet = {}

                # saving text of tweet
                parsed_tweet['text'] = tweet.text
                # saving sentiment of tweet
                parsed_tweet['sentiment'] = self.get_tweet_sentiment(tweet.text)

                # appending parsed tweet to tweets list
                if tweet.retweet_count > 0:
                    # if tweet has retweets, ensure that it is appended only once
                    if parsed_tweet not in tweets:
                        tweets.append(parsed_tweet)
                else:
                    tweets.append(parsed_tweet)

            # return parsed tweets
            return tweets

        except tweepy.TweepError as e:
            # print error (if any)
            print("Error : " + str(e)).encode('utf-8')

def main():
    # creating object of TwitterClient Class
    api = TwitterClient()
    # calling function to get tweets
    tweets = api.get_tweets(query = 'Donald Trump', count = 1000)

    # picking positive tweets from tweets
    ptweets = [tweet for tweet in tweets if tweet['sentiment'] == 'positive']
    # percentage of positive tweets
    print("Positive tweets percentage: {}
%".format(100*len(ptweets)/len(tweets))).encode('utf-8')
    # picking negative tweets from tweets
    ntweets = [tweet for tweet in tweets if tweet['sentiment'] == 'negative']
    # percentage of negative tweets
    print("Negative tweets percentage: {}
%".format(100*len(ntweets)/len(tweets))).encode('utf-8')
    # percentage of neutral tweets
    #print("Neutral tweets percentage: {} % ".format(100*len(tweets - ntweets -
ptweets)/len(tweets)))

    # printing first 5 positive tweets
    print("\n\nPositive tweets:").encode('utf-8')
    for tweet in ptweets[:1000]:
        print(tweet['text']).encode('utf-8')

    # printing first 5 negative tweets
    print("\n\nNegative tweets:").encode('utf-8')
    for tweet in ntweets[:1000]:
        print(tweet['text']).encode('utf-8')

if __name__ == "__main__":
    # calling main function
    main()
```