Kevin Zhang
2/25/2016
Software Design

Mini Project 3: Text Mining and Analysis of Twitch Chat

For this project, I decided to look into Twitch Chat and, using Twitch API, Pattern, and DSU, find out what the most common words were that used in a particular streamer's chat, as well as how these messages in the chat would be rated in a sentiment analysis. I used to watch Twitch a lot, and for those who know about it, Twitch Chat is about as troll and ridiculous as the worst of Reddit, maybe even more stupid. Yet still hilarious. Especially when messages are being spammed so fast you can barely comprehend what you're seeing, I decided to make an analysis program that would allow for users to see what was going on in the chat they were in, and just how bad and troll the chat really was.

In terms of implementation, there were mainly four parts to my program: the Twitch API, the reading and writing to the file, as well as updating everything in real time, the DSU which found the histogram of top 10 words, and the Pattern.en usage with sentiment to find the positivity/trollness of the chat. Looking first at the Twitch API, I mainly read up on the API and how to use it, but things got really confusing, so I ended up using the base code of an open source github script that created a Twitch bot in the chat. Using this code, I modified such that it gave me access to the Twitch chat of my choice, and then printed the messages out into terminal in real time.

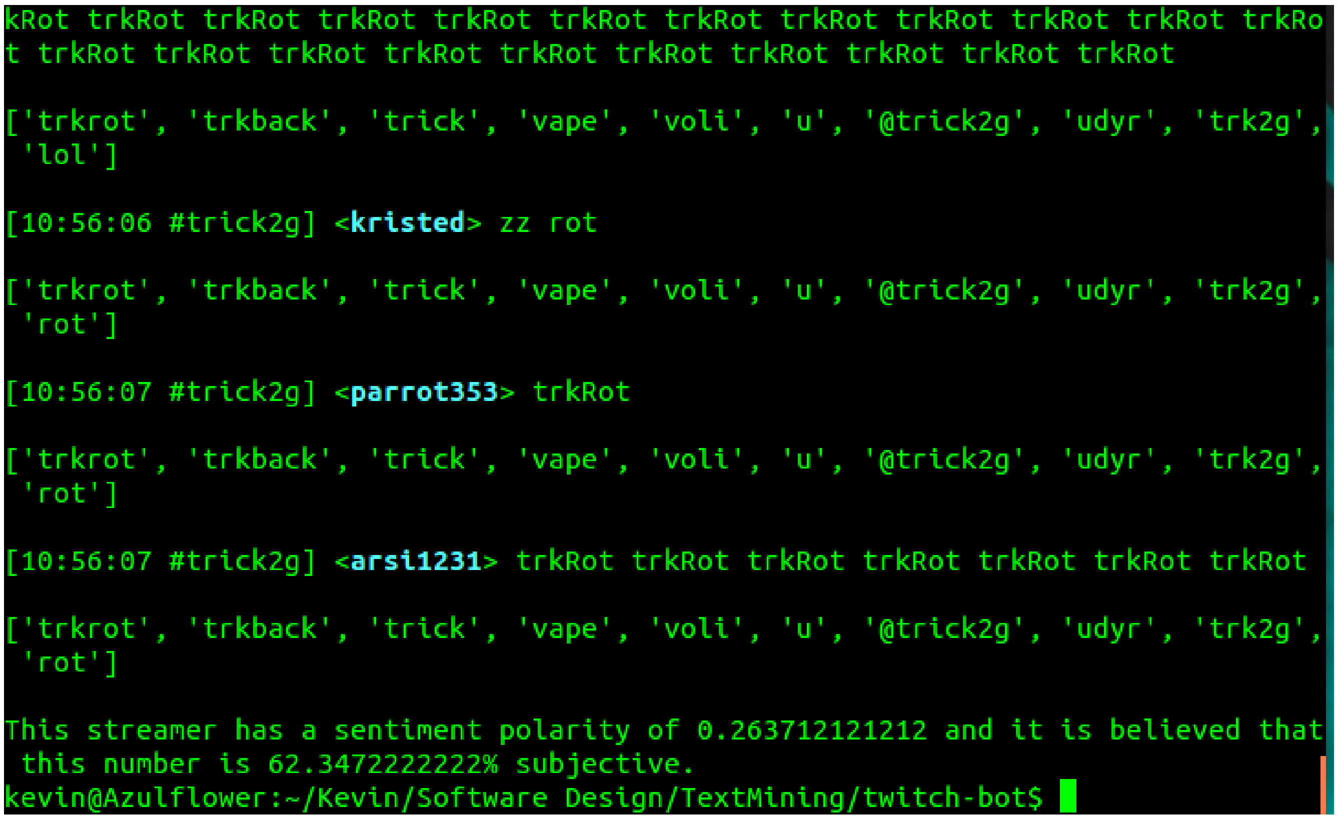
After that I implemented the histogram top 10 words using the DSU method we learned in class. I created a new script called textmining.py which had a method called analyze and took no parameters. It would open the file that had the chat history, read its entire contents, filter out common words, and then use the Decorating, Sort, and Undecorating strategy to find the top 10 words. There was heavy use of lists and tuples in this part of the program. A dictionary was also used to keep track of word frequencies. The result of this function was to print a list containing the top 10.

Then I decided to connect the two parts together, first thinking about pickle. I would open the file on the first run, and then in the loop I would iterate through a lot of dumping, closing, and reopening. The problem was that apparently pickle is a bit more complex and picky, as Twitch chat often uses emotes that have really strange unicode values. This broke my code many times, so in the end I decided to just use write and read which were more broad. Efficiency didn't suffer much, so I was okay with the substitution. There also implementation of the sys module, where I kept a counter incrementing until it hit 100, which meant 100 messages, and then it would sys.exit() and safely close the file and the script. To integrate the analyze function, I imported it as a module and then called it in the loop , making it such that the top 10 words would be updated in real time with every message that came in.

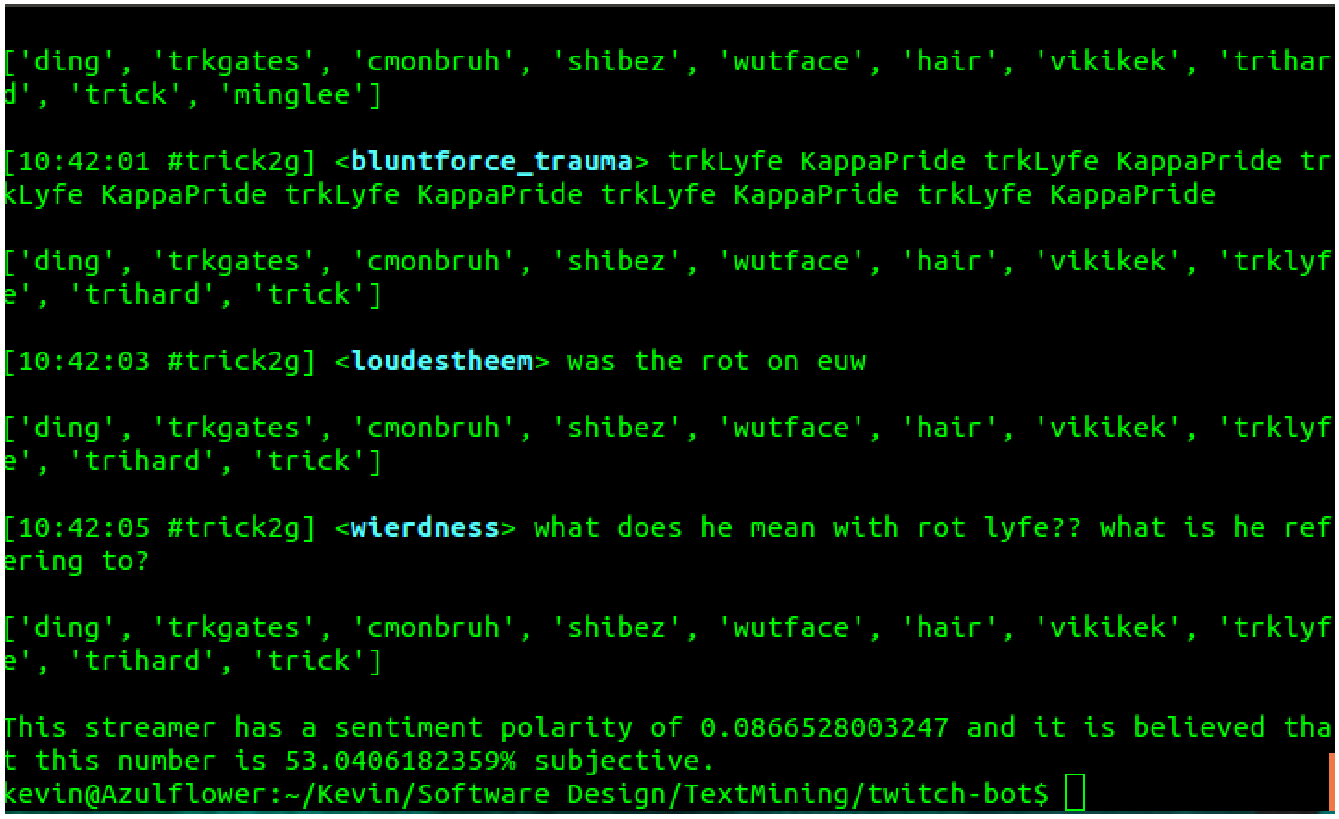
Finally, after everything was up and running, I decided to go one step further and implement sentiment analysis, just for kicks to see how bad the chat was. This was relatively easy, as basically I just made a function called sentiments that took in the entire contents of the file and then performed Pattern's sentiment on it. Integrating it consisted of adding in some code once the counter hit 100 but before the system exited to analyze the file once all the data had been compiled.

As for results, here are a few runs of my program on a popular streamer on Twitch name Trick2G, a League of Legends player. (Note that since it was 4 AM in the morning, this was the only streamer I had where I could get good data, I don't actually approve of what happens on his stream)

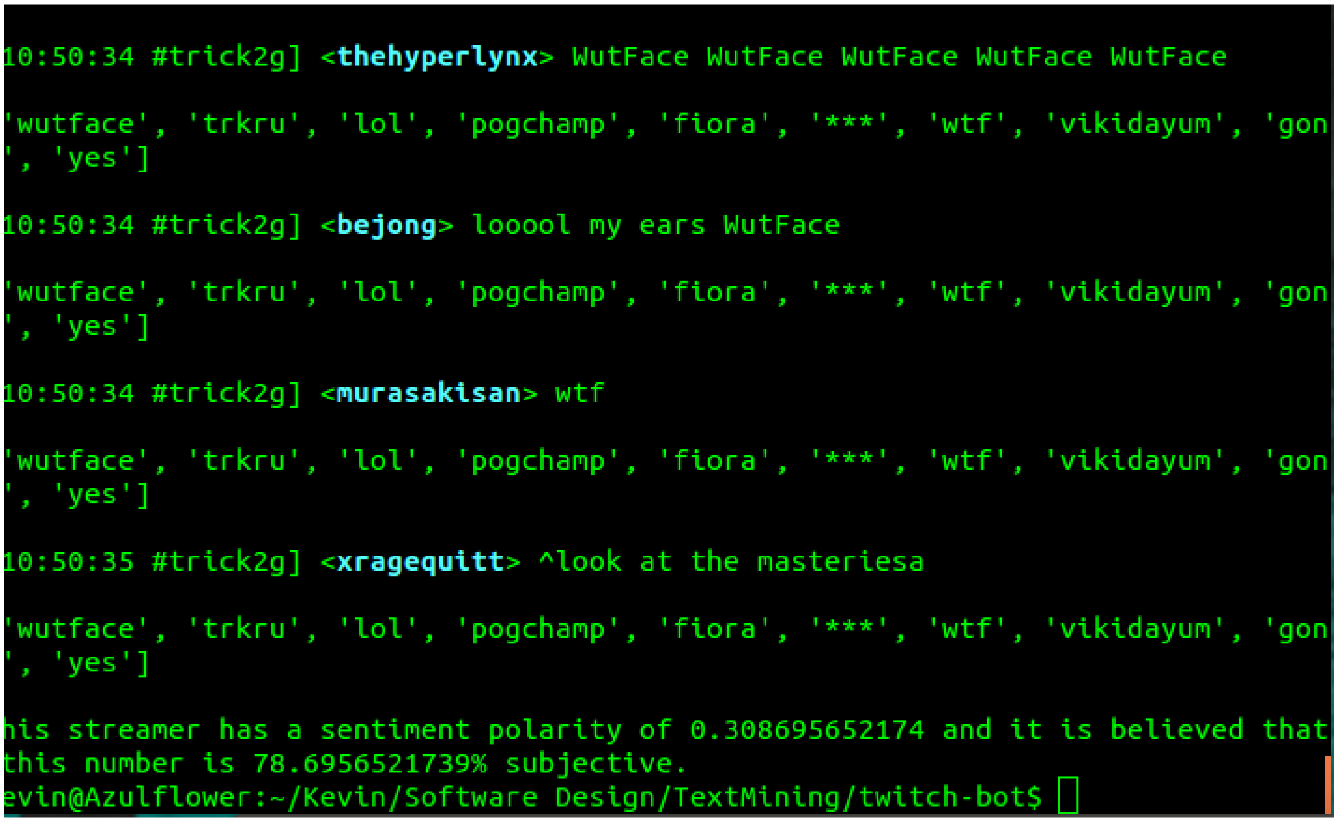
(Results on next page)



```
kRot trkRot trkRot trkRot trkRot trkRot trkRot trkRot trkRot trkRot trkRot trkRo
t trkRot trkRot trkRot trkRot trkRot trkRot trkRot trkRot trkRot trkRot

['trkrot', 'trkback', 'trick', 'vape', 'voli', 'u', '@trick2g', 'udyr', 'trk2g',
 'lol']

[10:56:06 #trick2g] <kristed> zz rot

['trkrot', 'trkback', 'trick', 'vape', 'voli', 'u', '@trick2g', 'udyr', 'trk2g',
 'rot']

[10:56:07 #trick2g] <parrot353> trkRot

['trkrot', 'trkback', 'trick', 'vape', 'voli', 'u', '@trick2g', 'udyr', 'trk2g',
 'rot']

[10:56:07 #trick2g] <arsi1231> trkRot trkRot trkRot trkRot trkRot trkRot trkRot

['trkrot', 'trkback', 'trick', 'vape', 'voli', 'u', '@trick2g', 'udyr', 'trk2g',
 'rot']

This streamer has a sentiment polarity of 0.263712121212 and it is believed that
 this number is 62.3472222222% subjective.
kevin@Azulflower:~/Kevin/Software Design/TextMining/twitch-bot$
```



```
['ding', 'trkgates', 'cmonbruh', 'shibez', 'wutface', 'hair', 'vikikek', 'trihar
d', 'trick', 'minglee']

[10:42:01 #trick2g] <bluntforce_trauma> trkLyfe KappaPride trkLyfe KappaPride tr
kLyfe KappaPride trkLyfe KappaPride trkLyfe KappaPride trkLyfe KappaPride

['ding', 'trkgates', 'cmonbruh', 'shibez', 'wutface', 'hair', 'vikikek', 'trklyf
e', 'trihard', 'trick']

[10:42:03 #trick2g] <loudestheem> was the rot on euw

['ding', 'trkgates', 'cmonbruh', 'shibez', 'wutface', 'hair', 'vikikek', 'trklyf
e', 'trihard', 'trick']

[10:42:05 #trick2g] <wierdness> what does he mean with rot lyfe?? what is he ref
ering to?

['ding', 'trkgates', 'cmonbruh', 'shibez', 'wutface', 'hair', 'vikikek', 'trklyf
e', 'trihard', 'trick']

This streamer has a sentiment polarity of 0.0866528003247 and it is believed tha
t this number is 53.0406182359% subjective.
kevin@Azulflower:~/Kevin/Software Design/TextMining/twitch-bot$
```



```
10:50:34 #trick2g] <thehyperlynx> WutFace WutFace WutFace WutFace WutFace

'wutface', 'trkru', 'lol', 'pogchamp', 'fiora', '***', 'wtf', 'vikidayum', 'gon
', 'yes']

10:50:34 #trick2g] <bejong> looool my ears WutFace

'wutface', 'trkru', 'lol', 'pogchamp', 'fiora', '***', 'wtf', 'vikidayum', 'gon
', 'yes']

10:50:34 #trick2g] <murasakisan> wtf

'wutface', 'trkru', 'lol', 'pogchamp', 'fiora', '***', 'wtf', 'vikidayum', 'gon
', 'yes']

10:50:35 #trick2g] <xragequitt> ^look at the masteriesa

'wutface', 'trkru', 'lol', 'pogchamp', 'fiora', '***', 'wtf', 'vikidayum', 'gon
', 'yes']

his streamer has a sentiment polarity of 0.308695652174 and it is believed that
this number is 78.6956521739% subjective.
evin@Azulflower:~/Kevin/Software Design/TextMining/twitch-bot$
```

The lines that look like they logging something are the actual messages from users on Trick2g's chat. The lists below that is the top 10 list that is being updated in real time with every message. The sentence at the bottom is the sentiment analysis part.

So in general, Twitch Chat was as bad as I expected: The top words were almost always either vulgar, something ridiculous and troll, or an emote. This is pretty typical of Twitch Chat, so there wasn't much to be expected there. The sentiment analysis also wasn't super surprising, as it basically always computed a low sentiment polarity rating, meaning negative connotations (or in this case, probably just really troll and people messing around). What did surprise me was the confidence that Pattern's sentiment had in its calls. Given that the second part indicates how subjective the analyzer believes the words are, I surprised that the subjectivity was so high. It could be that emotes and really weird Unicode values might throw it off, but even then it was interesting to note that the sentiment analyzer actually thought that the chat had some possible meaning in its endless spamming of drivel. Maybe Twitch Chat is something more than meets the eye? That answer lies in some future research that probably uses more sophisticated measures than mine.

There were a couple things that came to my realization during the making of this program. One was the idea that the results of the script could be skewed tremendously by one person literally spamming the same word over and over and over, which is actually very common in Twitch Chat. Another thing is that the chat is heavily influenced by what goes on in the stream, so if something really funny happened or the streamer just did something really cool, then for the next 5 minutes or so the chat will just explode in words and emotes in response to that particular event. However, aside from those things, the chat was basically random and had a very mob-like mentality. The slightest thing could tip off a massive chain of copying and pasting, resulting in a ton of the same words at some times, or maybe just a bunch of random words at times if the chat is in a lull and people are just mindlessly typing stuff. Also, since Twitch Chat is something that happens endlessly and without any sort of stopping mechanism, my program would have to be running 24/7 in order to get the best results. My examples only come from a history of 100 messages, which would be about the equivalent of the length of time for some big event to happen and the chat explodes. But in general, I think that my results captured at least one aspect of Twitch Chat: It is very, very troll.

In terms of reflection, I think that my understanding of the material and having the resources readily available to me made this project much easier. Time was a bit of an issue, as QEA is very hard and thus made my life miserable (this is being typed currently at 6AM in the morning). I felt like I could've dived deeper into this if I had the time, maybe go for visual graphics or using Markov synthesis. Pickling was also a major challenge, and maybe something that I could've spent more time on and perhaps gotten to work. I have done the Project Toolbox so I do fully understand how it works. I felt like my project was properly scoped given the limited time I had to work on it. I'm really busy right now, and finding the time to really dive deep and focus on the things I want to do is hard. Unit testing was hard as I used a Twitch API, and time was limited, but I did test each individual piece, such as testing that I could get messages to display using the API, testing my analyze function with random files, testing my sentiments function in a similar manner, and then testing pickling. Doctests weren't written in the interest of time, but rest assured that planned development as well as unit tests were used in the making of this program. I felt like I've learned a lot about web scraping and text mining, as well as a lot about how to analyze text in a variety of different ways. I also learned more APIs and sentiment analysis with machine learning. I will use this experience to improve my planned development, as well as plan better time management. My hunger for knowledge in computer science will never end, so this will probably be another stepping stone in my quest for more experience.