Trump Tweet Semantics Report

Luke Ireland

Table of Contents

1. Introduction

This project focuses on comparison of Sentiment Analysis algorithms and various other forms of Natural Language Processing(NLP), with a chosen dataset of Donald Trump's tweets.

Sentiment Analysis is classifying text into various classifications. In this case, it is into 3 classes: negative, neutral and positive. Natural Language Processing is the subfield of linguistics and computer science that looks at how computers process and analyze speech and text. Sentiment Analysis is a form of NLP.

Analysis methods include: - Frequency Distribution - Word Clouds - Sentiment Analysis - Tweet Generation

Frequency Distribution is another form of NLP that looks at the frequency of words and phrases within a given corpus. It is useful to be able to remove

stopwords (words that don't have any sentiment) from a corpus, and to highlight which words are more important in deciding the general sentiment of a given corpus, or the specific sentiment of a given sentence or paragraph.

Word Clouds provide visual representations of the frequency of words and phrases within a given corpus.

Tweet Generation typically involves using Markov chains, to replicate the prose within a corpora, and use that understanding to generate tweets that have the same prose as the original corpora.

Markov chains are models that describe the sequence of possible events, and their possibilities are dependent on the state set in the previous event.

The output from these analysis methods will be presented as a web page via Flask, which is a web framework for Python, and PonyORM, which is a object relational manager for Python.

1.1. Background

I was interested analyzing Trump's tweets for sentiment due to his controversial nature. It would be interesting to see, given what the media often say about him, if he truly is a bad (negative) person via unbiased (relative to humans, who are emotional and often irrational) sentiment analysis, and letting a machine look just at the words used, rather than the character of the person saying them.

I had also imagined it would be entertaining to analyse his tweets in other was, such as word clouds, frequency distribution and tweet generation, to see if he is overly repetitive of words or phrases, such as "MAGA" or "Make America Great Again", and to see if I could make tweets that are very similar to his style, in order to see if it's possible to capture (part of) his cognitive function in a Python function.

While looking at the various algorithms available to perform this sort of task, I thought it would be worthwhile to evaluate which of them works best on tweets, which are typically informal, 280 character long posts submitted to Twitter.

1.2. Aim

The aim of this project is to perform various types of analysis on the language used in Trump's tweet to see if any interesting trends arise. Mainly, this project seeks to inform and entertain people interested in Trump, for either good or bad reasons.

I will also be evaluating the performance of sentiment analysis algorithms on their speed and accuracy.

1.3. Objectives

1. Perform frequency distribution on variable length phrases.

2. Render word clouds of phrases.
3. Analyse whole tweets for sentiment.
4. Create unbiased classifer to initally label dataset
5. Find sentiment classifer algorithms
6. Use analysers in comparison to see which is the most accurate and the fastest.
7. Create tweets using data from Trump's Twitter account.

2. Literature Review

2.1. Guide (To-Do)

- Technology Justifications
- Background Reading
- Justify using anything
- Be critical of decisions/guides/opinions

I decided to use Python as it's my strongest language, plus it's flexibility across platforms and level/variety of API support makes it an obvious choice.

I originally planned to use Twitter's API via Twitter Search[1], but I couldn't use it due to being unable to apply for a Twitter Developer Account. I instead opted for someone else's collected tweets at Trump Twitter Archive[2]. The export format wasn't great, as you had to wait a while for the page to compile all the tweets into the correct format (When it would be useful to have it precompiled) and the page doesn't actually give you a JSON file, just a text output in JSON format, that you have to slowly copy and paste into a file and use programs to format the JSON into readable format.

I saw guides such as Basic Binary Sentiment Analysis using NLTK[3], Text Classification using NLTK[4] and Creating a Twtter Sentiment Analysis program using NLTK's Naive Bayes Classifier[5] using NLTK's Naive Bayes Classifier, but they used pre-processed data meaning I can't use them for my tweets. This guide[6] used Google's Natural Language API to perform Sentiment Analysis but this method required internet access, so wasn't particularly fast.

Eventually, I fell upon this article[7] which used TextBlob to perform sentiment analysis instead. TextBlob is a simplified text processing library for Python, and provides a simple API for performing Natural Language Processing tasks,

[1] https://github.com/ckoepp/TwitterSearch
[2] http://www.trumptwitterarchive.com/archive
[3] https://towardsdatascience.com/basic-binary-sentiment-analysis-using-nltk-c94ba17ae386
[4] https://pythonprogramming.net/text-classification-nltk-tutorial/
[5] https://towardsdatascience.com/creating-the-twitter-sentiment-analysis-program-in-python-with-naive-bayes-classification-672e5589a7ed
[6] https://www.freecodecamp.org/news/how-to-make-your-own-sentiment-analyzer-using-python-and-googles-natural-language-api-9e91e1c493e/
[7] https://www.geeksforgeeks.org/twitter-sentiment-analysis-using-python/

such as speech tagging, noun extraction, classification, translation and, most importantly, sentiment analysis.

I needed more methods of sentiment analysis, so I decided on using LSA, Random Forests (used as part of LSA), XGBoost[8], Logistic Regression and Multilayer Perceptron models to compare to my Naive Bayes classifier.

After coming across this article on algorithm comparison[^10], I found that creating a tf-idf transformer to use on the initial bag of words model massively boosts accuracy.

When implementing my models, I discovered that the fairest, most reproducible method of comparison was using Scikit Learn's Pipelines[^10], and began altering my code to minimise the difference between how classifiers are ran, to isolate the performance of the classifier down to the algorithm itself and not any pre-processing.

For tweet generation, I used Markovify[^11], which I found from this[^12] article attempting the same thing. The article listed multiple approaches, including using a Keras API and k-means clustering to build a Machine Learning model to feed into tweet generators, but that added a significant layer of obscurity to getting truly random tweets each time random tweets are requested. For example, it made it possible to get tweets about Hillary Clinton and North Korea in the same tweet/sentence.

3. Parsing

I used Python's JSON library to load the .json file into the program as a dict.

4. Tweet Cleaner

I removed non-alphabetical characters from the tweet.

5. Sentiment Scoring

I used TextBlob to retrieve a sentiment score

6. Sentiment Classification

I decided to choose rather narrow ranges for each sentiment class.

- Less than -0.25 = Negative
- Between +0.25 and -0.25 = Neutral
- More than +0.25 = Positive

[8] https://medium.com/@mc7968/whatwouldtrumptweet-topic-clustering-and-tweet-generation-from-donald-trumps-tweets-b191fccaffb2

7. Word Cloud

Here is a word cloud I created using the Python library wordcloud. Figure 1*Figure 1* - WordCloud of phrases of length 4.

8. Frequency Distribution

I used NLTK to look at the most common words and phrases of different lengths.

9. Tweet Generator

I created my own class using Markovify[9], that would force the generator to generate tweets containing a given user word.

10. Web Application Presentation

I used PonyORM to...

11. Evaluation

Results and findings

12. Conclusion

13. References

[9]https://github.com/jsvine/markovify