A Comparison Between Polling Data and Social Media Sentiment – 2020 US Presidential Election

This research project involved a thorough comparison between poll ratings and social media sentiment for the two primary candidates across four states in the lead up to the 2020 US Presidential Election.

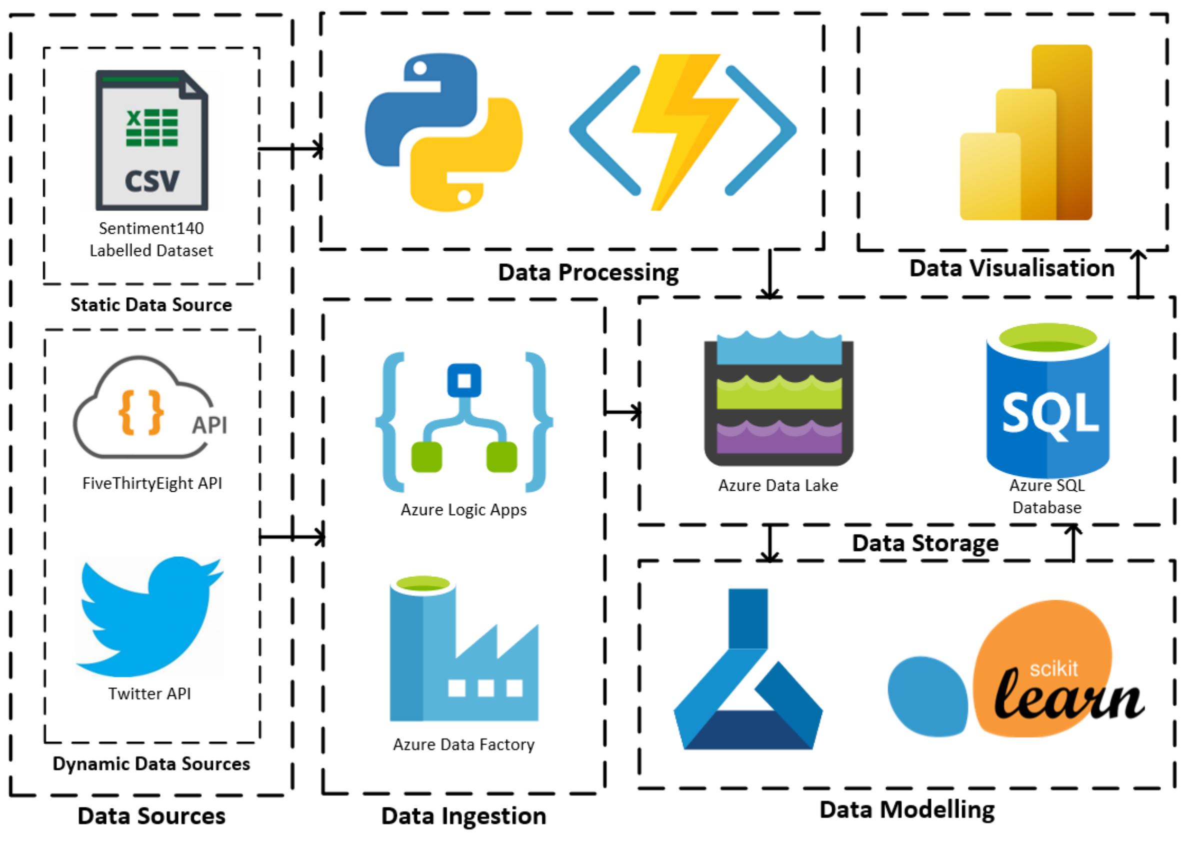
Introduction

The 2020 US Presidential Election was held on 3rd November, with the lead up to the election particularly emotive and divisive. In 2016, the polling companies suffered a shock when they failed to foresee Donald Trump beating Hillary Clinton and winning the race to the White House. I was keen to understand the disparity that exists between polling data and the sentiment on social media in the lead up to the 2020 election.

Over the 10 weeks prior to the vote I collected tweets about the two primary presidential candidates, Joe Biden and Donald Trump. For the same period I collected aggregated polling data. I also wanted to understand how the disparity between social media sentiment and polling levels varied by state. I collected this data across four states, two that voted Republican in 2016 (Florida and Texas) and two that voted Democrat (California and New York).

Using a pre-labelled twitter dataset, I trained a sentiment classification model to predict the sentiment of a tweet based on the text it contains. I applied various text pre-processing and feature extraction steps to the scraped tweets, before applying my trained sentiment classification model to extract their sentiment. I then compared the sentiment of these tweets with the equivalent polling levels, and produced a dashboard to visualise the results.

The diagram below shows, at a high level, the architecture of my project.



I will now walk through the different stages involved in the project - Data Sources, Data Ingestion, Data Processing and Data Modelling. I will then present the final results in the form of a dashboard.

Data Sources

This project required three main data sources: the Twitter API, an API for polling data, and a dataset of pre-labelled tweets.

I used the Sentiment140 dataset [1] of 1.6 million labelled tweets from Kaggle. This dataset was generated from the Twitter Search API, with tweets labelled as positive or negative based on the sentiment of emoticons contained in them rather than being human annotated. The information contained includes the tweet, user, tweet id and sentiment of the tweet - either positive or negative.

The Twitter API is free to use, and allows you access to different tools and endpoints to interact with the Twitter platform. There are a few steps that must be completed in order to access the API [2]. You must first apply for a developer account, for which you will need to supply information about your planned use of the API. In my case, it took roughly two weeks for my account to be approved. You then receive your access keys and tokens that need to be saved. You are then able to access the API, and the Twitter developer website has a number of guides and documentation for how to connect and use the different endpoints.

There is a huge amount of polling data that exists in the United States, conducted by a variety of different organisations. It is hard to find a single polling source that is likely to be accurate, so for that reason I used an aggregated polling service. FiveThirtyEight [3] don't usually commission polls themselves, instead aggregating the numerous different polls that exist to produce an overall picture. They include data on a state level, as well as at a national level. The data is publicly available on their website, with the raw data available through GitHub [4] which can be easily accessed through a REST API.

Data Ingestion

I manually downloaded the Sentiment140 dataset through Kaggle's website. I used Microsoft Azure services to extract data from the two API's at periodic intervals over a 10 week time period in the lead up to the election.

Azure Data Factory was used to ingest data from the Twitter API on a periodic basis. I used Twitter's Search API to search for tweets from two topics – 'Trump' and 'Biden'. I extracted tweets from four different states – California, Texas, Florida and Texas. I searched for tweets from each topic-location combination on a daily basis, extracting 100 tweets in each search. The search was limited to a 24 hour period preceding the API request to ensure the tweets were from that day. The data was saved to Azure Data Lake storage in a Hierarchical folder system.

To extract the polling data, I used an Azure Function written in Python. The Function ran a GET request to the raw polling data hosted on the GitHub of FiveThirtyEight, taking the

‘Presidential Poll Averages’ csv file. I filtered this data, only keeping the averages from the locations I was focussing on. I had to do some cleaning on the dates, as well as removing columns that were not needed. The Azure Function was scheduled to run on a weekly basis, with the data again saved to Azure Data Lake Storage.

Data Processing

Before developing a sentiment classification model, the text data needed to be processed into a suitable form for modelling. The data processing involved a series of normalisation, noise removal and features extraction steps. These steps were used to prepare the labelled twitter data for modelling, as well as to develop a processing pipeline to be run on the newly scraped Tweets.

The research papers [5] and [6] provided inspiration for the processing of social media data. The processing pipeline I developed included the following steps – extraction of emoticon sentiment, the removal of Twitter specific language, conversion of all text to lowercase, acronym expansion, punctuation removal, stopword removal and finally lemmatisation.

I firstly looked at emoticons contained in Tweets. I created an emoticon dictionary, consisting of all of the most commonly occurring emoticons and their sentiment (either positive or negative) from [7]. For each tweet I searched for the presence of any emoticons contained in the emoticon dictionary. I created a count of the number of both positive and negative emoticons for each tweet, and extracted these into separate columns. I then removed the emoticons from the tweet text.

The next step involved removing all Twitter specific language including urls, hashtags, targets (@) and RT’s. I then removed all remaining punctuation from the text, before converting all text to lowercase for standardisation purposes.

Following the idea in [5], I created an Acronym dictionary. I created a Python script to scrape all of the acronyms on the No Slang [8] website. The output file includes a column containing the acronym and another containing the expanded acronym. I then looped through each tweet and expanded any acronyms that exist.

I used NLTK’s built in stopwords function in Python to remove all stopwords in the text. The SpellChecker Python library was used to correct any potential spelling mistakes in the text. It should be noted that all of the spell checking functions I tested in Python take extremely long to run on a large dataset, and they don’t tend to have the greatest accuracy. The final step involved lemmatising the words to return them to their stems.

Due to the size of the dataset and the amount of pre-processing needed, with the limited compute power I had available a large amount of time would be needed to process all of the data. For this reason, I converted the Python code into a Cython script. Cython is an optimising static compiler, which gives you the combined power of Python and C allowing us to tune readable Python code into plain C performance by adding variable type declarations in the Python syntax.

After cythonizing the Python code, the only remaining function consuming significant time was the spell checker. As the function is written in Python it is difficult to speed up further. Excluding this function, it takes Python roughly 43.8160s to process 100 tweets. The equivalent code running in Cython takes only 0.2296s, almost a 150 times speed improvement.

Modelling

Once the text data had been processed it was ready to be modelled. I experimented with three different types of model on the labelled twitter dataset – supervised learning approaches, knowledge-based sentiment analysis models, and a deep learning approach.

For the supervised learning approaches I used a count vectorizer and TFIDF vectorizer to convert the processed text tokens into features that could be modelled. I tested the performance of four different supervised learning algorithms on the data – Logistic Regression, Naïve Bayes, Support Vector Machine and Random Forest. The table below shows the results of the supervised learning algorithms, trained on 75% of the labelled data and tested on the remainder.

Count Vectorizer	**Logistic Regression**	**Naïve Bayes**	**Support Vector Machine**	**Random Forest**
Accuracy	0.7292	0.7284	0.5456	0.6512
Precision	0.6939	0.6637	0.9486	0.5482
Recall	0.7378	0.7528	0.5240	0.6875
F1-Score	0.7152	0.7054	0.6751	0.6100
AUC	0.7298	0.7313	0.6330	0.6574
TFIDF Vectorizer	**Logistic Regression**	**Naïve Bayes**	**Support Vector Machine**	**Random Forest**
Accuracy	0.7348	0.7164	0.7112	0.6504
Precision	0.7282	0.6669	0.7251	0.5804
Recall	0.7300	0.7308	0.7034	0.6723
F1-Score	0.7291	0.6974	0.7142	0.6230
AUC	0.7347	0.7178	0.7114	0.6531

Of the four models used, the Logistic Regression and Naïve Bayes models showed the best performance. The Logistic Regression model using the TFIDF Vectorized features displayed the highest accuracy, F1-Score and AUC score of all the models tested.

The knowledge-based sentiment analysis approach involved using the SenticNet library [9]. SenticNet is a concept-level sentiment analysis tool that can be applied to tasks such as polarity detection and emotion recognition by leveraging the underlying sentiment of words, rather than training a model on co-occurrence counts. It provides a set of semantics, sentics and polarity associated with 200,000 natural language concepts.

The model developed using the SenticNet library involved splitting each Tweet into a list of words, before using the library to extract the sentiment value of each word. The SenticNet library assigns a score between -1 and 1 to each word. We then take the average sentiment of all words in the tweet. If this average is greater than 0 we assign a positive sentiment to the Tweet, otherwise a negative sentiment.

I then trained a classification model on three features – the SenticNet calculated sentiment of the text, as well as the number of positive and negative emoticons. The results of this model are shown in the table below.

	SenticNet Model
Accuracy	0.6013
Precision	0.8690
Recall	0.5629
F1-Score	0.6833
AUC	0.6442

From the results of the knowledge-based model, we can see it generally performs significantly worse than the supervised learning approaches above. Due to the nature of knowledge-based models, they struggle to deal with context, as well as concepts such as sarcasm and double negatives.

The final approach used was a deep-learning model. I used a Recurrent Neural Network (RNN), a type of neural network used for processing sequence data. They can process a sequence of vectors by applying a recurrence formula at every time step. The particular type of RNN used was a Long-Short Term Memory (LSTM) model.

An embedding layer was used in the model to extract features from the word tokens. I used the GloVe embedding, which is a count-based embedding model that is trained on word-to-word co-occurrence counts. The idea behind this embedding is that we expect a greater co-occurrence between two words that occur together more often. It has shown better results in many tasks than the Word2Vec embedding, although more recent approaches such as BERT have displayed even better performance.

The results of the LSTM model are shown below:

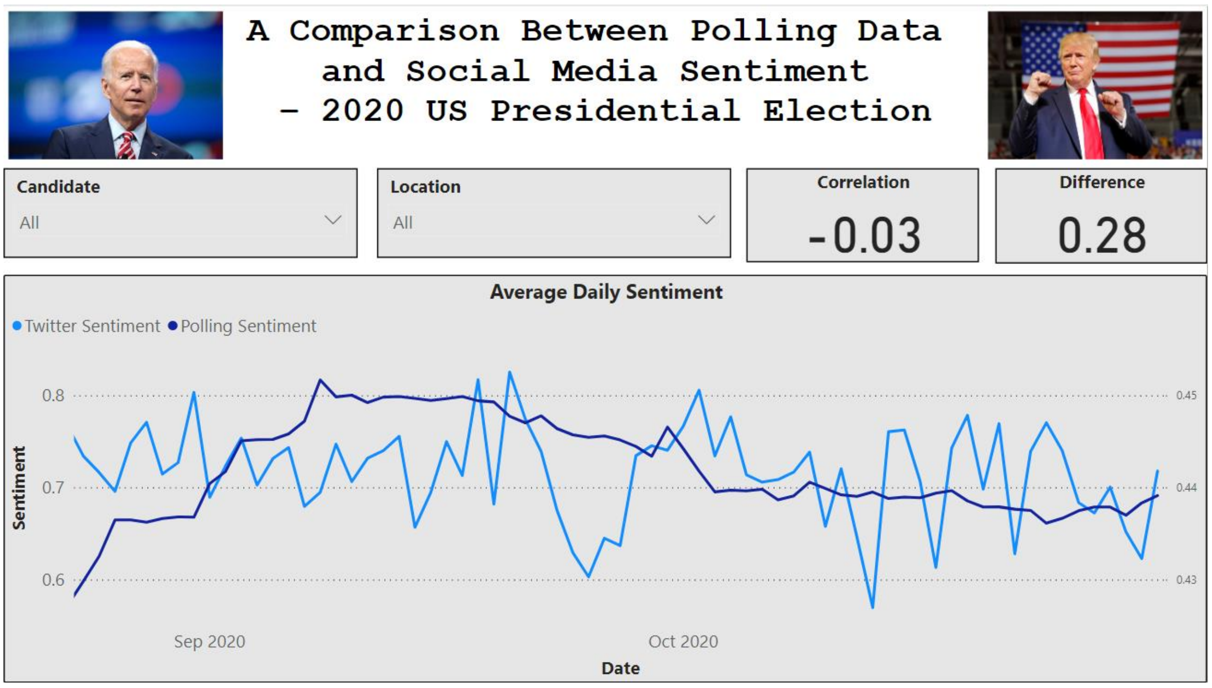
	LSTM Model
Accuracy	0.6409
Precision	0.8387
Recall	0.6010
F1-Score	0.7002
AUC	0.6671

We can see that whilst the results of our deep learning approach are better than that of the knowledge-based model, the Logistic Regression model still shows superior performance.

Evaluation

The Logistic Regression supervised learning model was used as the final model. I saved the trained model so that it could be used later on. I created a Python script to concatenate the scraped Twitter csv files, and ran the data processing pipeline and saved model on these Tweets to predict their sentiment from the text they contained.

I imported this data into Power BI, along with the Polling data. I created a dashboard to visualise the results, which is shown below.



A Comparison Between Polling Data and Social Media Sentiment – 2020 US Presidential Election

Candidate: All

Location: All

Correlation: -0.03

Difference: 0.28

The line chart provides a time-series view of the daily average Polling levels and Twitter sentiment over a period of 10 weeks prior to the election. The left sided axis relates to the average sentiment, and the right sided axis to the average polling value. The two slicers enable us to filter the data by both candidate and location. The data cards display the correlation and difference between the Twitter sentiment and Polling levels.

The immediate thing to notice is the large difference between the social media sentiment and polls, with an average difference of 0.28. This difference is fairly consistent no matter the candidate/location combination. The reason for this difference is likely due to the difference in what is being measured. Sentiment can take a value anywhere between 0 (negative) and 1 (positive). The polls tell us the % of respondents who are likely to vote for each candidate, therefore with two primary candidates they must have an average value close to 0.5. It is therefore a lot more useful to look at the relative differences between each, hence the reason multiple axis's were used.

Just focussing on the relative changes to both Twitter sentiment and Polls, it is still very difficult to see much of a pattern between them. The correlation value of -0.03 supports this, informing us there is virtually no relationship between them. This is an extremely unexpected result - as the sentiment on Twitter for each candidate changes we would expect the polling data to support this.

A final observation is the difference in variance. We can clearly see that Twitter sentiment fluctuates significantly more than the Polling data. This is likely explained by the fact that social media sentiment is likely to rapidly change in reaction to events, whilst voters are only likely to change their opinion on a very rare basis.

Conclusion

Collecting more Tweets may help to give a better picture of the overall social media sentiment at each point in time. Collecting 100 tweets per candidate per day per location is useful, but only gives us a snapshot of overall sentiment on Twitter. A larger sample size would be needed to produce significant results.

The results of the sentiment analysis models on the pre-labelled dataset were disappointing, with the best model achieving an accuracy of 73%. Part of the reason for this may be down to the quality of the dataset - the tweets were not human annotated, instead labelled just based on the emoticons they contained. I would also like to implement a deep learning approach using the BERT word embedding to improve the model further.

References

[1] - Go, A., Bhayani, R. and Huang, L., 2009. Twitter sentiment classification using distant supervision. CS224N Project Report, Stanford, 1(2009), p.12. (https://www.kaggle.com/kazanova/sentiment140).
[2] - https://developer.twitter.com/en/docs/twitter-api/getting-started/guide
[3] - https://fivethirtyeight.com/features/polls-policy-and-faqs/
[4] - https://github.com/fivethirtyeight/data/tree/master/pollster-ratings
[5] – Agarwal, A., Xie, B., Rambow, O., et al., 2011. Sentiment Analysis of Twitter Data.
[6] – Kharde, V., Sonawane, S., 2016. Sentiment Analysis of Twitter Data: A Survey of Techniques.
[7] – https://en.wikipedia.org/wiki/List_of_emoticons
[8] - https://www.noslang.com/dictionary/
[9] - https://sentic.net/