Sentiment Analysis of Tweets

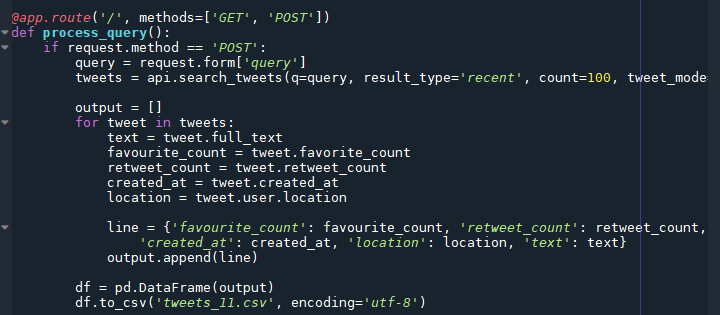
Introduction

In this report I will outline how I used a number of different tools to determine the sentiment of a tweet from the social media website Twitter. I will begin by looking at the process used for gathering the tweets needed for the project before describing how the tweets were processed to make them more suitable for the sentiment analysis tools that I used. I will then proceed to examine how each of the different analyses, using TF-IDF, Vader and neural networks, performed at assigning the correct sentiment for the tweets.

Gathering and Processing of Tweets

I began by signing up for a Twitter developer account which provided me with the authorisation needed to use Twitter’s API to download the tweets and all of the metadata for each tweet. The downloading process was facilitated by using tweepy which, when used in conjunction with the API, allowed me to download a large number of tweets at one time. This stage was performed using a Flask web application in Python. As can be seen from the screenshot included below I downloaded 100 tweets per topic and extracted a number of variables from the json output that was returned, saving these variables into a pandas dataframe and then into a csv file. I ended up using tweets from 10 different topics, giving a total of 1000 tweets to begin with.

I was able to remove exact duplicate tweets easily by using the drop\_duplicates function for a pandas dataframe. For retweets which had been added to slightly and for ‘quote tweets’ I had to decide on an individual basis if the retweet was significantly different from the original tweet. I also deleted tweets which were purely descriptive or informational, such as those from a TV channel promoting an upcoming TV show. After this process I was left with just 423 tweets. Although we were advised to share tweets amongst our class group this did not appear to be happening so I decided to continue with just my own batch of tweets. It would obviously have been beneficial to the project to have more information to use in the analyses but, given the time that it was taking to go through each topic in the screening process, I felt that my number of tweets would have to do.

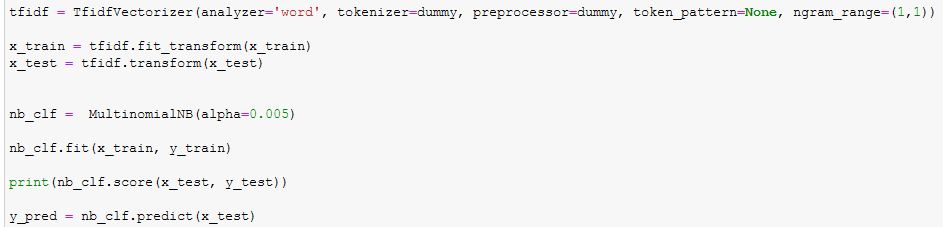


*Fig. 1: Flask app code snippet*

The next step involved deciding what sentiment was the dominant one being expressed in a tweet, using the labels ‘positive’, ‘negative’ and ‘neutral’. For this project I decided to choose topics which might elicit a strong reaction, whether positive or negative. I ended up with 194 ‘negative’ tweets, 143 ‘positive’ tweets and 86 ‘neutral’ tweets. Before moving on to the actual analysis, the body of each tweet was trimmed so that punctuation marks and basic words such as ‘the’ and ‘this’ were removed. Longer versions of words or plural forms of a word were also trimmed to return the core word. I also used the regex package to remove all pieces of text that started with ‘http’ as there were a lot of links within the tweets. These steps were taken so that the body of a tweet would be more lean and the sentiment would be more likely to come to the fore during analysis. Finally I separated the ten tweet subjects into three separate broader topics; the politics topic contained tweets using the search words “Varadkar”, “Sinn Fein” and “Qanon”; the TV topic contained tweets using the search words “Eastenders”, “Tommy Tiernan” and “Eoghan McDermott”; and the others topic, which was an assortment of subjects, contained tweets using the search words “Shamrock Rovers”, “Pancakes”, “Burren” and “Daniel Kinahan”.

TF-IDF

The first tool used to analyse the tweets to determine their sentiment was the Tfidf Vectorizer package from the scikitlearn library. The data was split into test and train sets and a tfidf.fit\_transform operation was carried out on the training tweets, as seen in the code snippet below. Multinomial naive bayes was used with my sentiment label as the output variable (y\_train in the figure below).



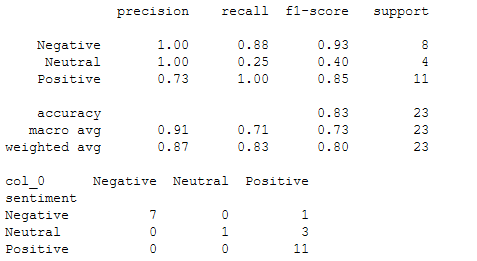
*Fig. 2: Performing tweet analysis using Tfidf and Naive Bayes*

This analysis was performed for the three topics individually as well as all together. The best result was found when using the tweets with the topic of TV. I also tested each topic three times with the n\_gram parameter set to just test individual words, to also include bigrams, and to also include both bigrams and trigrams. The results generally improved as the ngram\_range was increased, however, for the politics set of tweets the accuracy actually dropped from 62% to 58% with the inclusion of bigrams before rising to 69% with the further inclusion of trigrams.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Tfidf*** | **Politics** | **TV** | **Others** | **All** |
| **Unigrams** | 62 % | 70 % | 51 % | 55 % |
| **+ Bigrams** | 58 % | 83 % | 62 % | 65 % |
| **+ Trigrams** | 69 % | 83 % | 65 % | 66 % |

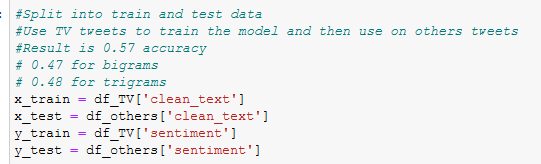
*Fig. 3: Table of Accuracy of Tfidf analysis per topic*

As can be seen from figure 4, the results of the analysis using the tweets with the topic of TV are good. In particular, all 11 ‘positive’ tweets were correctly labelled as such by Tfidf. For this particular topic a lot of the tweets were very clearly positive or negative in their tone, thus making them clearly distinguishable. The relatively poor results achieved by the ‘others’ tweets is possibly explained by the fact that they contained a number of disparate topics which may have made it more difficult for the model to establish a definite set of trends or indicators to be used in classification.



*Fig. 4: Results of analysis on ‘TV’ tweets, ngram\_range(1, 3)*

Finally, I also performed a number of analyses where I trained the model on one topic of tweets and then used that model to attempt to correctly classify another topic. Overall, accuracy here was quite poor with the figure at around 40% quite frequently. Interestingly, while the addition of bigrams and trigrams boosted the accuracy for models which were performing poorly it tended to decrease the accuracy level if the model was already quite accurate.



*Fig. 5: Tfidf – Training on one topic and testing on another*

Vader Sentiment Analysis

Using the Vader package to determine the sentiment of the tweets was quite straightforward given that the tweets had already been prepared. I performed the Vader analysis on the same cleaned tweets, again separating the tweets into three separate topics. The tweets with the topic of ‘TV’ were again most accurately classified, with an accuracy rate of 63%. Since I was only dealing with unigrams in this instance, the figure can be compared with the Tfidf result for ngram\_range(1,1), which was 70%.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Tfidf*** | **Politics** | **TV** | **Others** | **All** |
| **Accuracy** | 62 % | 70 % | 51 % | 55 % |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Vader*** | **Politics** | **TV** | **Others** | **All** |
| **Accuracy** | 45 % | 63 % | 59 % | 56 % |

*Fig. 6: Results for Tfidf vs Vader*

As can be seen from Figure 6, Vader performed particularly poorly compared to Tfidf in the classification of political tweets. However, most of these tweets were labelled as ‘negative’ and the Tfidf and naive bayes model predicted that almost all tweets from this class would be negative. Given the small sample size and the skewed nature of the data it appears that the Tfidf model had difficulty in correctly identifying any non-negative tweets. The overall accuracy for both methods was very close.

Neural Network Classification

Finally, I trained a model to try to correctly label sentiment using neural networks built with keras and tensorflow. The model included an embedding layer, a LSTM (long short term memory) layer and a dense layer. The dense output layer consisted of three outputs representing positive, negative and neutral labels. It also included dropouts between the layers to avoid overfitting. As can be seen from figure 7, the results were generally quite similar to those achieved using Tfidf and Vader. It is interesting to note though that here the highest accuracy was achieved when all tweets were combined into one set. This is possibly explained by the fact that neural networks tend to improve their performance when the amount of data they are given is increased.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Neural Network*** | **Politics** | **TV** | **Others** | **All** |
| **Accuracy** | 64 % | 58 % | 38 % | 70 % |

*Fig. 7: Results for Neural Network analysis*

Conclusion

In this project I used a number of techniques in an attempt to correctly classify as negative, positive or neutral a set of tweets. The processing of the tweets themselves was actually the most time-intensive part of the project, and if I were starting again I might try to automate some of the cleaning and labelling of the tweets rather than going through them individually myself. It was seen that the results of the analyses were not brilliant, but some of the models did achieve reasonably high accuracy scores. In particular, the model using Tfidf returned quite good results, especially when bigrams and trigrams were included in the model. The neural network also had a decent accuracy score when trained on all of the tweets. Given the complex and at times subtle nature of language and the way we use it, it is not that surprising that the accuracy wasn’t better.