# NLP_Tweet_Sentiment_Analysis


image courtesy of Ady Teenagerinro and unsplash

## What's the Buzz?

Many companies want to keep track of public sentiment about various products. Positive sentiment might drive production for creators or inventory management for retailers. It might drive the attention of influencers, news organizations, and consumers. Twitter is a platform where individuals and companies express themselves and have conversations and is full of information about how people feel about particular products.

We four intrepid data scientists, as a project for our data science intensive bootcamp through Flatiron school, set out to create a model that could determine how people feel about products or brands through their tweets. Twitter is gold mine of emotional sentiment, as users impulsively express themselves in short, terse terms. Tweets about given products are easy to collect as users use hashtags to communicate the topics of their expression.

We created a model when given a tweet or series of tweets and a product would determine how the user felt about that product. This is trivial for a human to accomplish, but our model can do this for thousands or even millions of tweets in a small amount of time.

This is not a trival problem. Twitterspeak is rife with spelling errors, shortened words, acronyms, hashtags, and very specific words and proper nouns. These are hard or impossible for a model to learn and are even confusing for humans much of the time. Emotions are also complicated and a tweet may have a mix of emotions, ambiguous emotions, or conditional emotions. The spectrum of feeling is much greater than our categories of positive, negative, or neutral.

Emotions are also relative. Consier the phrase, "iPhones are better than Android phones." Is this a positive or a negative tweet? It depends! This is why we chose to give our model a clue to which product we want the emotions to be classified in reference to. We used the product name supplied in the 'product' column of our table to replace instances of that product in the tweet test with 'targetproduct'. This helped our model to contextualize emotions and deal with this relativity.

A model only learns as well as it's labels, and our data was labeled by humans. Each human may have a different interpretation of a tweet. The data was most likely labeled by different humans, and so there may be different opinions about the emotional context of a tweet expressed by different human labelers.

These considerations make this problem particularly sticky, and we are proud of the success that our models had in decifering the tweets. As is often true with NLP problems, the structure of the model is often less important than how the data is prepared.

We tried many different approaches to this problem, from recurrent neural networks to convolutional neural networks, to a simple logistic regression model. Each used model approached the problem in a different way, but they all were able to classify over 3/5ths of tweets correctly, giving a client a general sense, over a large sampling of tweets, what the sentiment is about their product in the twitterverse.

# The Data

We trained our model using 4 datasets hosted on data.world's Crowdflower:

- The Brands and Product Emotions dataset
- The Apple Twitter Sentiment dataset
- The Deflategate Sentiment dataset
- The Coachella-2015 dataset

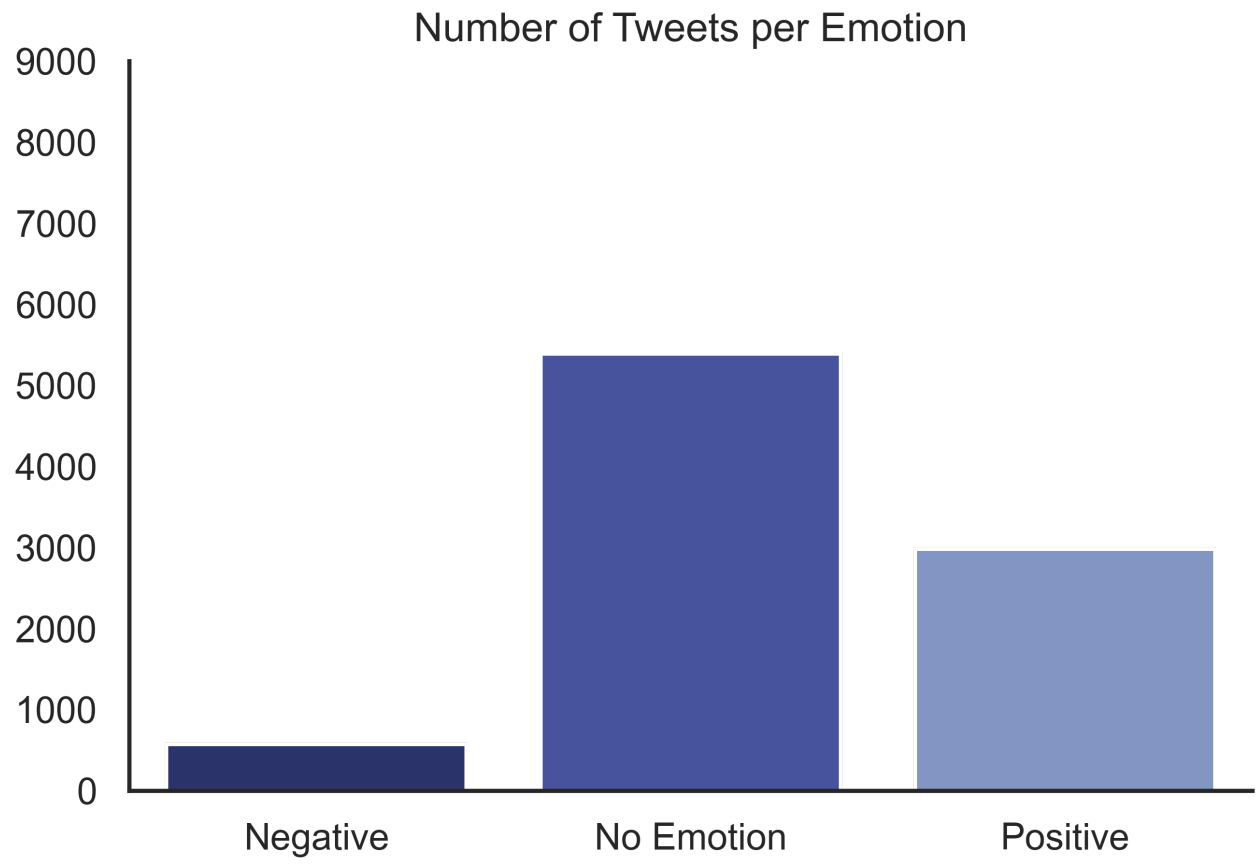
# Data Exploration

| Index | Tweet Text | Tweet Target | Tweet Sentiment |
|---|---|---|---|
| Each row is one tweet | Contains the text of the tweet | Contains the target the tweet is directed towards | Contains the sentiment of the tweet, can be positive, negative, or none |

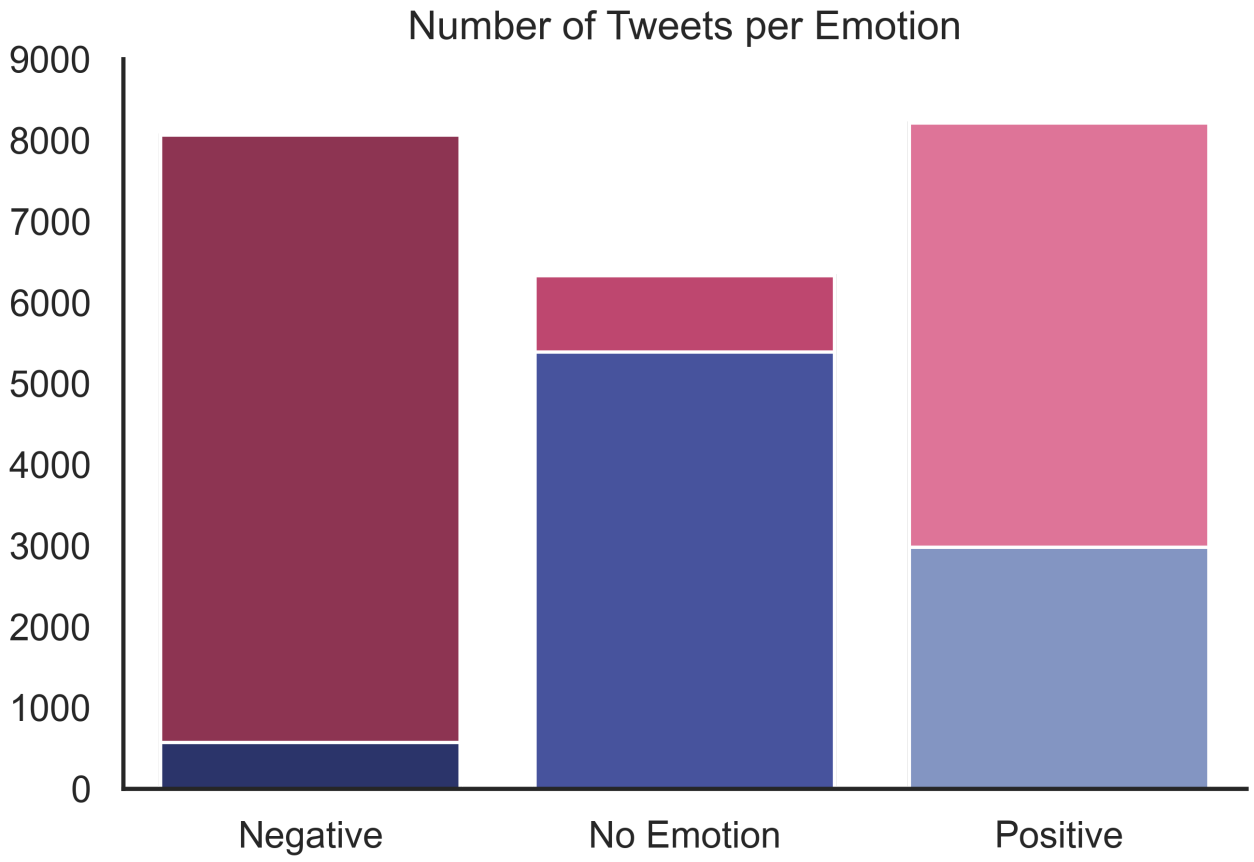
Our base data, which is the Brands and Product Emotions dataset, comes with three columns with categorical values, with every row being a separate tweet. The first column contains the text of the tweet. This text is copied striahgt from the original source, so it contains all the little unique things that tweets such as '@' mentions and '#' hashtags. The second column contains the target of the tweet. The target can be subjects such as 'iPad', 'iPhone', or 'Google' to name a few, and they denote what their respective tweet's subject is. For example, if the tweet target is 'Android App', we can assume that the tweet text in the same row has something to do with it. The last column contains the tweet sentiment, or the type of emotion the tweet text is showing. There are three possible values: positive, negative, and no emotion. A positive value would symbolize that the tweet has a positive feeling towards their listed target, while the opposite would be true if it was negative. A value of no emotion would mean that the tweet does not have a particularily strong feeling towards either side. As the tweet sentiment is the value that we are trying to predict, this will be our target column in our predictive models.

# Class Imbalance

To ensure that our model is accurate, we'll have to check the class disparity of our targets. As we are setting the tweet sentiment as our target, we must check how many of each different target type that we have.



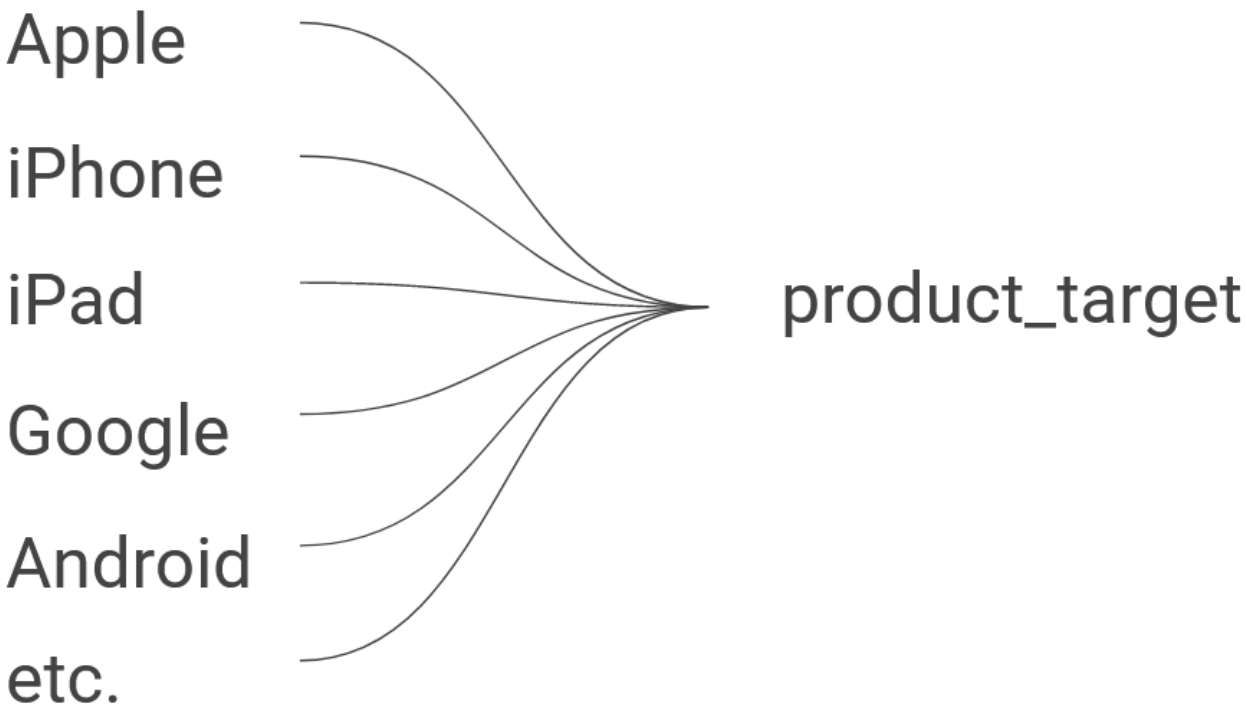
As seen above, the class disparity is quite severe. Ignoring the fact that the 'no emotion' values are almost double the amount of 'positives', the number of 'negative' sentiment tweets is far too low in order to properly class the tweets. This is bad as it could lead to our model learning to get a high accuracy score simply by guessing 'no emotion' for any input. As we do not want our model to overfit, we must deal with it. While oversampling our 'negatives' or undersampling our 'no emotion' tweets is possible, we decided to instead import more data from similar datasets, which are datasets that must 1. contain rows of tweet texts and 2. contain a variable relating to the tweet sentiment. We found three extra datasets that fit our requirements, the Apple Twitter Sentiment dataset, the Deflategate Sentiment dataset, and the Cochella-2015 dataset. After we imported the extra data, processed them to match our base dataset, and concatenated them together, our class disparity becomes the following:




With the introduction of our new data, the difference in the number of each class becomes much smaller. Both 'positive' and 'negative' tweets now have a similar amount, while 'no emotion' tweets follow closely behind. Thus we do not have a need to undergo any over or undersampling.

# Feature Engineering

Our main goal for our project is to create a model that will predict sentiment relative to a flexible product. To do this, we do not want our model to make predictions based on user's sentiments based on prior tweets, but instead to judge the tweet solely on the tweet sentence content. In other words, we do not want our model to naturally predict tweets about iPhones to be positive when we have a lot of existing tweets with users that sing praises about their new iPhones. To do so, we will deal with it by replacing any instances of a tweet's target in their respective text with one, all-encompassing phrase "product_target". Thus instead of saying "I love my new iPhone.", it will become "I love my new product_target.". By doing this our model will be more focused on the "I love my" portion of the sentence istead of whether or not existing data supports positive tweets for an iPhone. Not only does this help deal with overfitting, this would allow our model to work on targets outside of the ones in our dataset, such as if we wanted to look at tweets about a festival or a sports team.




# Data Cleaning and Preprocessing

As our tweet text comes straight from the source, they are undoubtedly very messy. We must first preprocess the data in order to convert them to a form safe for consumption by our predictive models. The following is a list of processes we took to turn our dirty source tweets into clean, filtered data:

1. Split the tweet into tokens
2. Convert all capitalized letters into lower-case

3. Remove punctuation
4. Remove twitter jargon such as @ mentions
5. Remove leftover numbers
6. Remove words with accents
7. Remove stop words
8. Replace instances of the target in the text with 'product_target'
9. Remove empty strings
10. Lemmatize the words
11. Rejoin all the tokens into one string

Here is an example of a tweet looks like after we undergo cleaning on it:


After going through every row and applying our cleaning function to it, we will drop our target column as we have no more need of it. Thus our resulting dataset after pre-processing will look like the follwing:

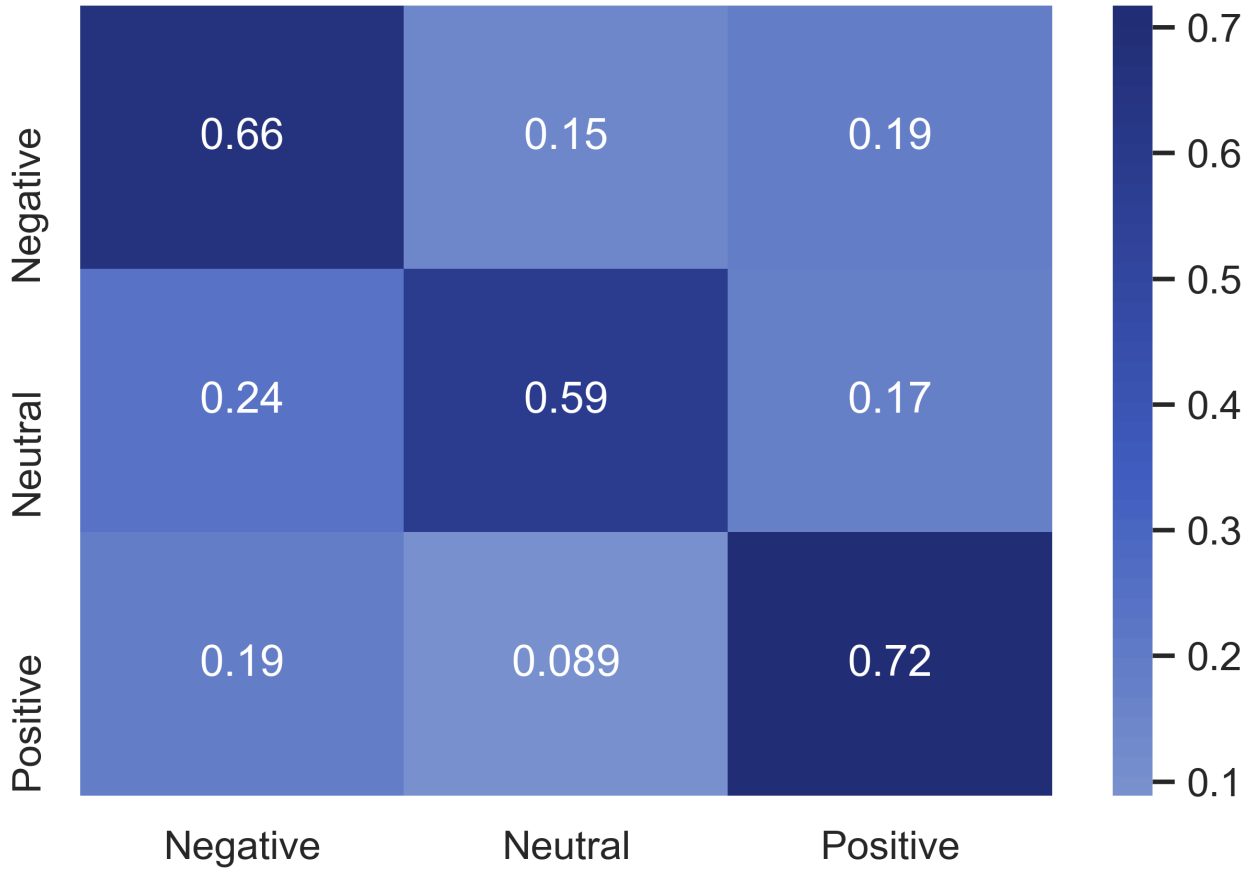
| Index | Tweet Text | Tweet Sentiment |
|---|---|---|
| Each row is one tweet | Contains the cleaned text of the tweet | Contains the sentiment of the tweet, can be 0, 1, or 2 |

# The Models

We explored 3 very different model architectures: shallow models, convolutional neural networks, and recurrent neural networks.

## Shallow Models

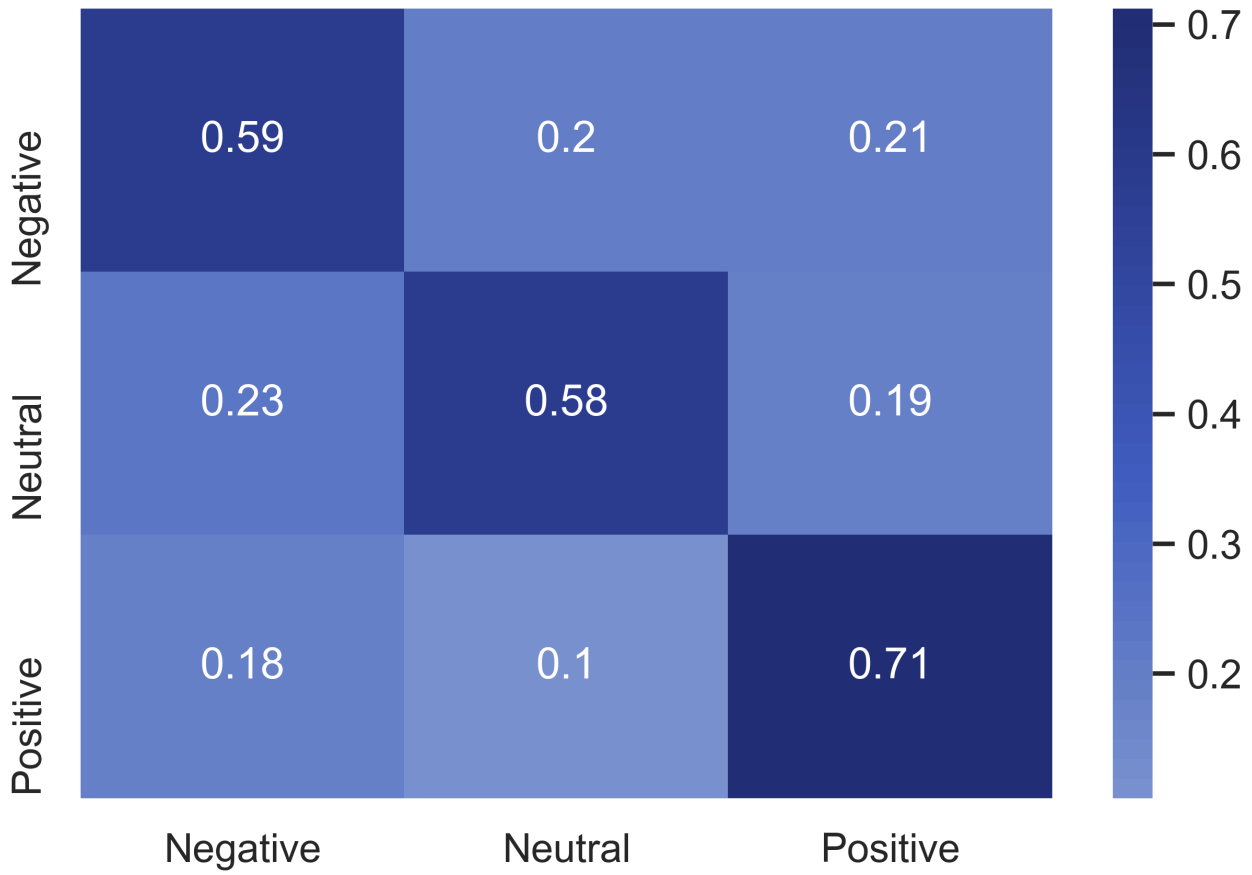
We ran 3 shallow models using Multinomial Naive Bayes, Logistic Regression and Random Forest Classifier, and one ensemble model with all three classifiers together. Count and TF-IDF vectorizers were used to preprocessed data sets. Pipeline and grid search were used to find the best hypertuning parameters for our models. Logistic Regression model performed better than other models with 66.0% prediction accuracy on Count Vectorizer in unigram range (1,2). It means the model predicting the best on a single word or two adjacent words. The biggest advantage of Logistic Regression is fast performing, a few seconds less than a minute comparing to several minutes, up to 40 minutes for the ensemble model.




## Convolutional Neural Networks

We also choose to try a convolutional neural network implementation with an embedding layer as this is a common modeling architecture in NLP and in particular with sentiment analysis. The final structure of the model we arrived at began with an embedding layer, followed by a 1 dimensional convolutional layer with dropout, followed by a max-pooling layer. We then flattened the output and passed it through two dense layers with dropout and ending with a 3 node dense layer with softmax activation for prediction output.

With this model we were able to achieve approximately 62% overall accuracy with this model, with a fairly balanced distribution of accuracy across all classes. The positive class was predicted a bit more accurately at 71%, but was still only about 10% more accurate than the least accurate class.



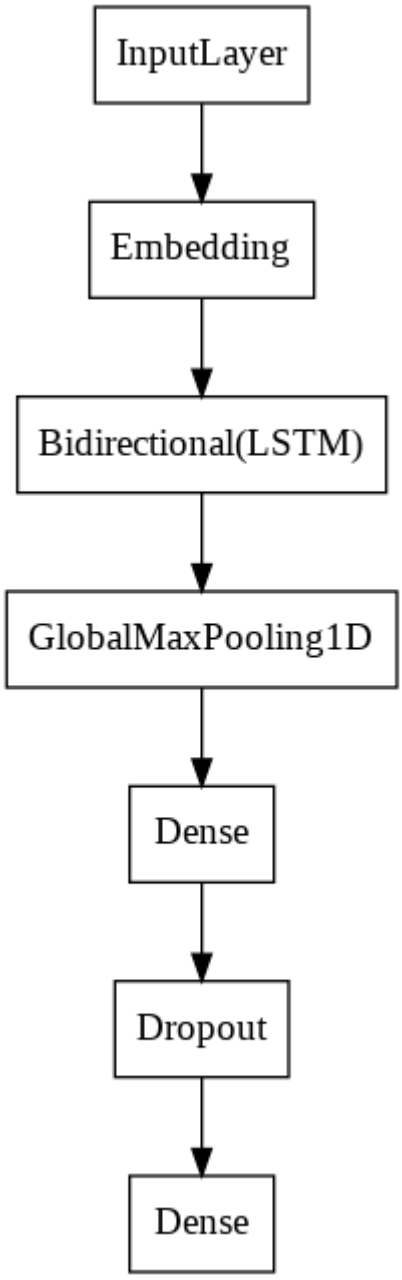

## Recurrent Neural Networks.

We began with a rather simple RNN using a trainable embedding layer followed by a long-short term memory layer, a max pooling layer to process the sequence data from the LSTM, a densely connected layer and finally a smaller dense output layer. This model performed reasonably well, achieving a 62% accuracy on the validation data.

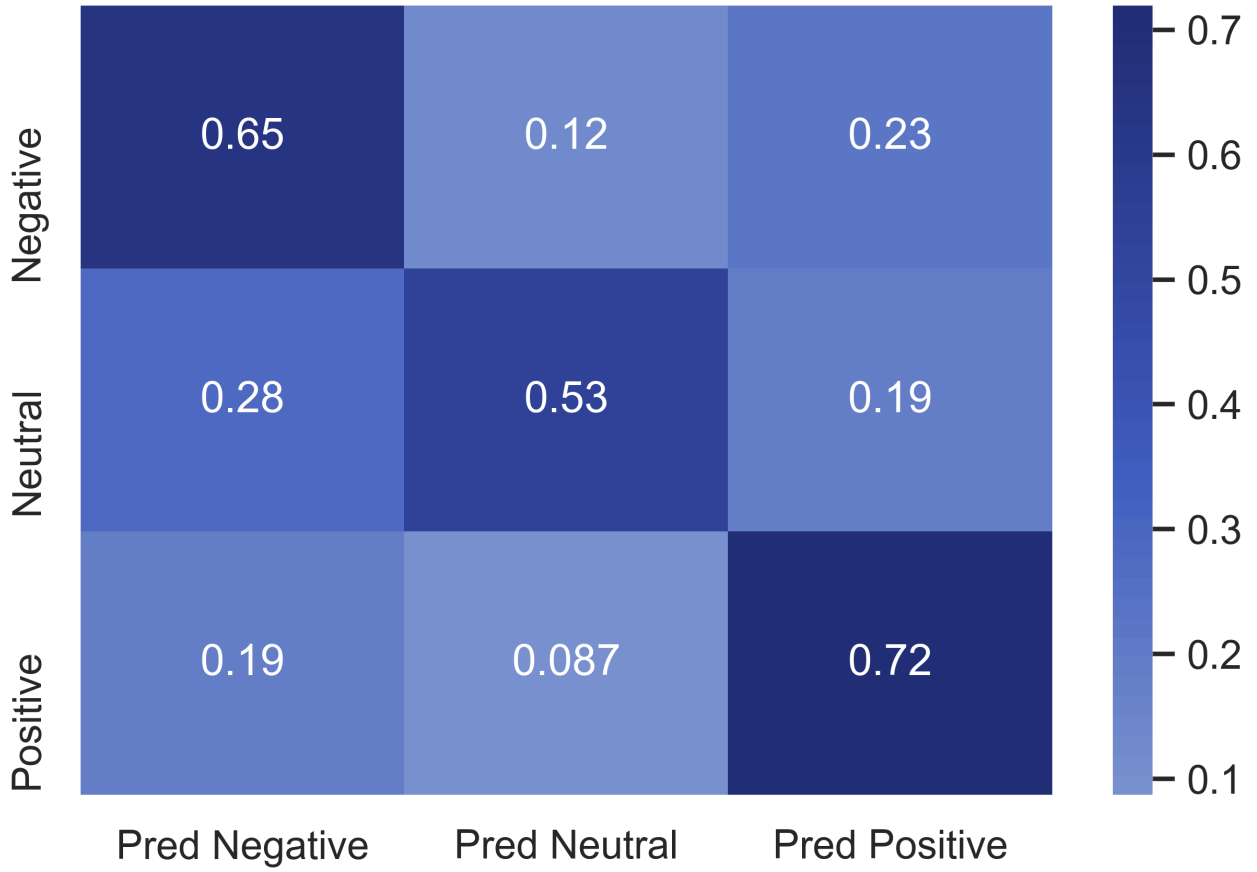
We also explored using pretrained GloVe Embeddings[1] rather than a trainable embedding layer, but because so much of the vocabulary from our dataset was nonstandard English, the GloVe dictionary, though massive woulc only encode about 2/3rds of it. The model lost too much information from the tweets to be able to perform well. We also tried initializing the embedding layer with the GloVe embeddings and then letting train from there, but this also gave disappointing results.

The change that did end up making some difference was to make the LSTM layer both smaller (from 25 neurons to 5 neurons) and to make it bidirectional so it read the tweets both forward and backward at the same time. The bidirectionality combined with the memory of the LSTM's memory cell allowed the model to read words both in the context of the words that came before them, but also in the context of words that came after. This is similar to how humans read, retroactively contextualizing what they have read based on what they read later.

The final LSTM model was actually simpler than the first simple model:



These changes raised the accuracy from 62% to 64.5% with a reasonable distribution of errors across classes. It did somewhat better on positive emotion labeled tweets, and tended to miscategorize neutral tweets as negative. This ended up being as much accuracy as we could squeeze out of this style of model.




If you want to learn more about this type of model, please refer to this excellent article by Raghav Aggerwal.

# Summary

We created a model that can label tweets about a specific product with 67% accuracy. This can be used, in conjunction with a twitter crawler to pick tweets related to a brand or product, to determine how positively or negatively a specific product is being regarded in the Twitterverse during a specific timeframe. The best model we made was also the simplest, fastest, and least computationally demanding, a logistic regression model. However we used a lot of preprocessing to achieve this success. We focused the model on the target product by replacing the product name with 'targetproduct', tokenized and lemmatized the words to boil them down into their basic meanings, and then vectorized them using a count vectorizer to index them by how commonly they show up. The model used this heavily processed version of the tweets to determine their emotional content.

# Going Forward:

Future researchers might explore better ways to lemmatize and prepare twitterspeak to be more semantically accessible to predictive models. They also might try using other kinds of neural networks, such a gated recurrent unit models or simpler densely connected feed forward models. While complex deep learning solutions may improve on our work, it's also possible that tweets can best be classified using 'bag of words' style modeling. In this case the specifics of word encoding might be what makes the difference and using GloVe embeddings to encode words for a shallow model could be productive.

# References:

[1]Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. [pdf] [bib]