**Project Plan**

**A.I.**

**SEIS 764**

Jideofor Onyeneho

Darius Nyaundi

Tony Blonigan

Sander Quint

Meshal Almuhareb

**Data Source Description**

The data was obtained from Kaggle (<https://www.kaggle.com/thoughtvector/customer-support-on-twitter>). The data contains tweets that were sent from and to customer support twitter accounts of several companies (Amazon, Apple, etc.).

**Number of Records**

Our data contains 2,811,774records.

**Number of Attributes**

Our data has 7 attributes.

**Attribute Descriptions**

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Type** | **Definition** |
| tweet\_id | Numerical | A unique, anonymized ID for the Tweet. Referenced by response\_tweet\_id and in\_response\_to\_tweet\_id. |
| author\_id | Numerical | A unique, anonymized user ID. @s in the dataset have been replaced with their associated anonymized user ID. |
| inbound | Categorical | Whether the tweet is "inbound" to a company doing customer support on Twitter. |
| created\_at | Timestamp | Date and time when the tweet was sent. |
| text | String | Tweet content. Sensitive information like phone numbers and email addresses are replaced with mask values like \_\_email\_\_. |
| response\_tweet\_id | Numerical | IDs of tweets that are responses to this tweet, comma-separated. |
| in\_response\_to\_tweet\_id | Numerical | ID of the tweet this tweet is in response to, if any. |

|  |
| --- |
| **General Statistics** |
| * The dataset contains around eleven years' worth of tweets (from 2008-2017) |
| * 54.7% of the tweets in the dataset are questions sent to customer service |
| * 45.3% of the tweets in the dataset are answers sent by customer service |
| * The dataset contains tweets written by 702777 different accounts |

**Tools**

|  |  |
| --- | --- |
| **Tool** | **Purpose** |
| Matlab | Used for data pre-processing, logistic regression, machine learning, AI, and prediction |
| Python | Used for data pre-processing , logistic regression, machine learning, AI, and prediction |
| R | Used for data pre-processing, logistic regression, machine learning, AI, and prediction |
| Tableau | Used for data visualization |
| PowerPoint | Used for presenting the final project |

**Methods**

* Natural Language Processing
* Word Embedding
* Attempted Word2Vec
* Convolutional Neural Network
* Recurrent Neural Network
* LSTM

**Problems & Questions**

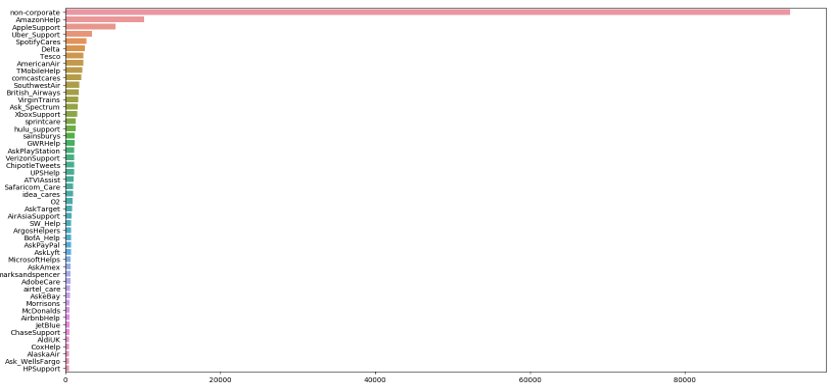
* Predict which company wrote tweet.
* Predict whether a tweet was inbound or outbound.
* Sentiment analysis on tweets.

**Pre-Processing**

The twcs.csv file was preprocessed into sequences for neural network modeling. First, all text in the columns was converted to lowercase letters and all non-alphabetical characters were removed. URL protocol headers (e.g. "https://") along with user name headers (e.g. "@seis764") were removed. After that, a tokenized document was created using words. Very common words often featuring 2 or less characters and rare words that were typically above 14 characters were removed. Stop words including "about" and "been" were also removed in the sentiment and the inbound-outbound models only. There are important stop words such as "et" for Eastern Time in the author classification model, so they were not removed in that model. In the inbound-outbound model, empty documents were removed from the training data. The words were lemmatized using parts of speech details for normalization. In the author classification model, Porter Stemmer was used to stem words. Some analysis tasks altered these steps slightly. Since we were completing several different tasks, we decided there was no reason to ensure that our preprocessing was exactly the same, because there would be no metrics to compare between models, anyway.

**Author Classification**

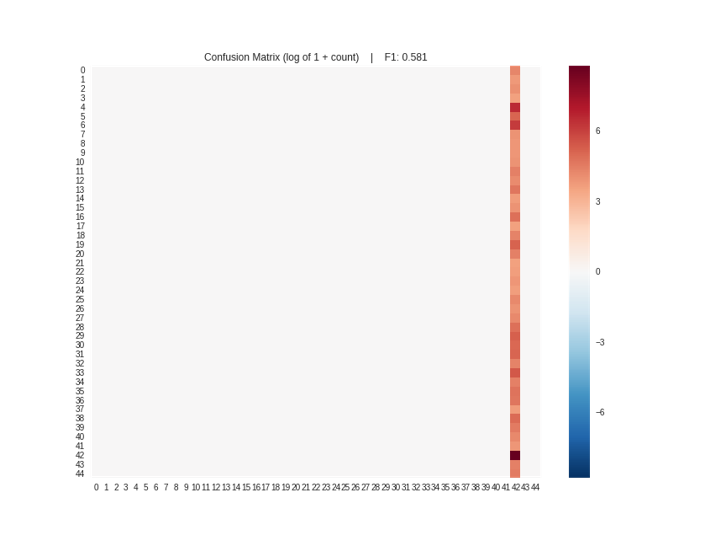
One way we use this data to explore artificial intelligence was setting up a classification problem where we use the text of a tweet to determine which author wrote the tweet. For this problem we did the following pre-processing steps, which differ slightly from the steps outlined above: Remove punctuation, removed @username tags, converted all websites two simple generic http://, all lower case, make sure each word was only single spaced, applied Porter stemming, tokenized words, applied post padding to the length of the longest sentence, developed word embedding using the class labels. also, since many of the tweets were written by individuals who only had made a handful of tweets we chose to combine all of the non corporate accounts into a single class we called non-corporate. This decision created a challenge in that it made the data set very unbalanced, as you can see in the histogram below.



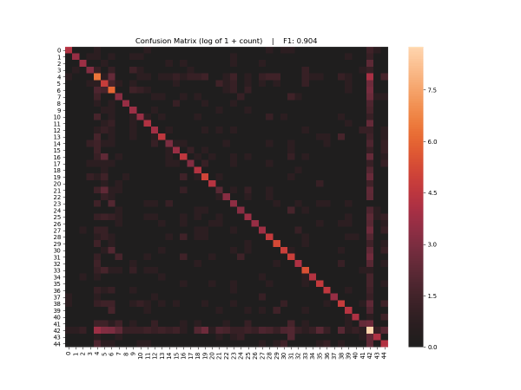
I should also mention that we simplified the websites and remove the @user tags because the problem seemed too trivial if we had left them in.

Since we had over two million tweets, when initially trying to train the training was taking like an hour for Epoch, so we decided to use a sample of 200000 tweets so that we could try more networks out.

The first architecture we tried was an RNN consisting of one embedding layer, 1-2 RNN layers, and 1-4 dense layers. We were able to achieve 58.1% F1 with these models, Regardless of the hyperparameter settings we used. For example, we tried using GRU and LSTM units in the RNN portion, and we tried several different amounts of hidden units in each layer, and we tried drop out and L2 regularization. But each Network would get the same F1. looking at the confusion Matrix, it is obvious that this is because it was classifying each example as non-corporate, he majority class.

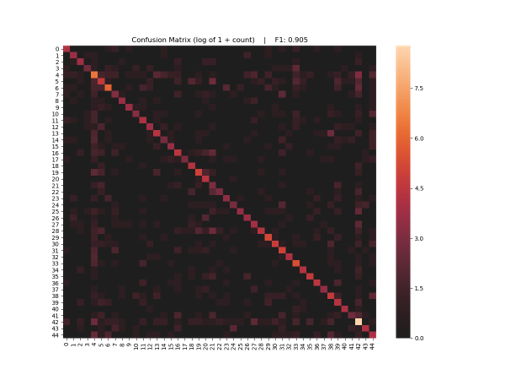


We suspected that the word embeddings might not be getting trained well on account of the depth of the neural network, and the fact that the RNN layers needed to back propagate Through Time. So next we decided to train a word embedding independently, hoping that the errors could be back propagated better in the shallow embedding Network, and then we could use transfer learning to apply those embeddings to a new, deep model. Several attempts were made to train word2vec models using negative sampling. But we add mini errors when trying to use the trained embeddings in a new model. So, we ended up using a simple word embedding model that had one embedding layer with a 500-dimensional representation of each word, Followed by a flat layer, and a 45-unit softmax classification layer. To our surprise this model achieved significantly better performance, even though we only hoped to get better word embeddings. This model was able to achieve an F-1 score of 90.4%, what we thought was fantastic. You can see in the confusion Matrix below that it is obviously much better, but still has some Miss classification, especially of the non-corporate and other classes with many examples.



Next, we tried using these word embeddings in an RNN model, hoping for even better performance. But surprisingly we were only able to replicate the performance of our previous l RNN Networks. Maybe we could have tried waiting the penalty by class so that there was a higher penalty for misclassifying the tweets that were not non-corporate, or tried downsampling the non-corporate class to get better results.

Finally, we tried a fully connected Network architecture. Does not work consisted of the embedding layer of flattened layer and then several dense layers separated by Dropout. We found that very high dropout, on the order of 50%, work the best. But even the best fully connected architecture only improved performance marginally,And we were only able to get an F1 score of 90.5%, which is probably a statistically insignificant compared to the word embedding only model. But, Although the F1 scores are very similar, you can see from the confusion Matrix that the errors this model makes are different from the errors in the simple word embedding model. Well the simple word embedding model misclassified disproportionately the majority classes, the Deep neural network seems to misclassify all of the classes more evenly.

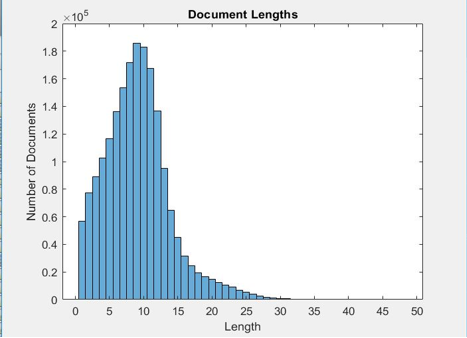


**Inbound vs Outbound Binary Classification**

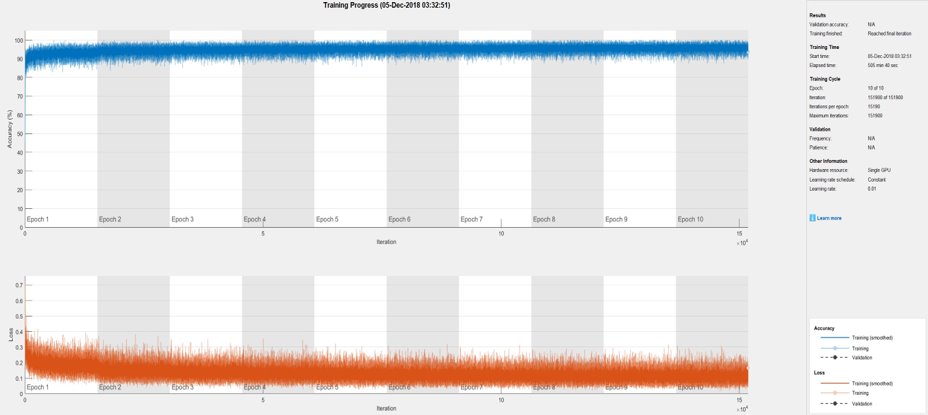
Classifying tweet type (inbound (Company account)/outbound (User Account)) from the tweet text was another way of exploring the data. There are many applications for this model, such as quickly deciding whether a tweet need to be looked at further or not.

The model was built using an input sequence layer, word embedding layer, LSTM layer and FC-SoftMax-Classification layers at the end for the classification task.

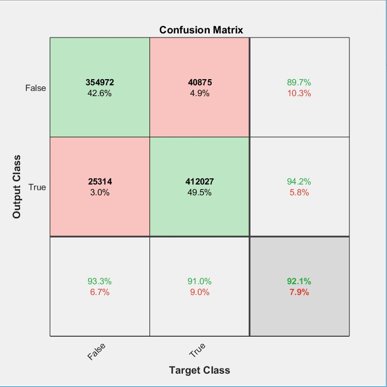
Input sequence length was decided to be 13 after plotting documents length.



The main challenge we faced building the model was resource limitation, as the model took over 8 hours on a single GPU to be trained.



The Model itself was performing well as the model Accuracy = 92%. The confusion matrix below contains more information about the model accuracy

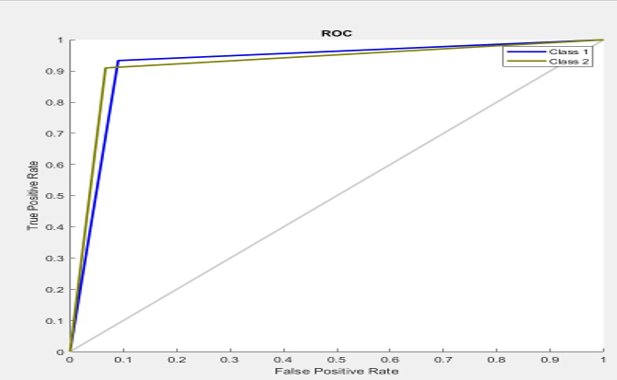


As showing in the confusion matrix above, the F1 Scores are:

F1 Score (False): 79%

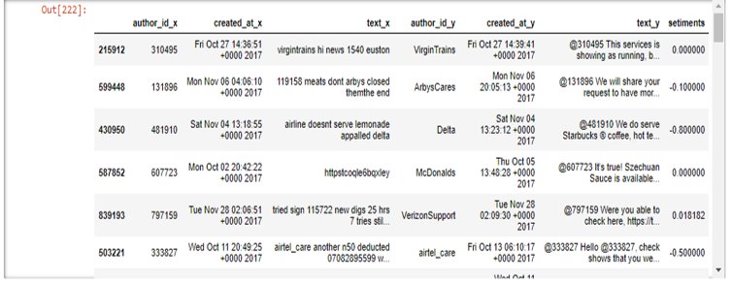
F1 Score (True): 76%

Plotting ROC shows the performance of the model by measuring the area under curve (AUC).

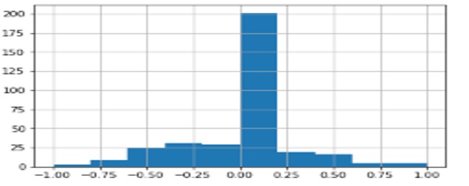


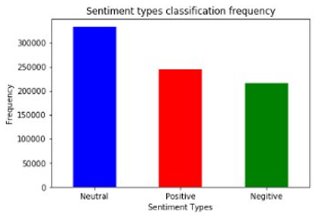
**Sentiment Regression**

We wanted to do sentiment analysis, but our data set didn’t include any sentiment related target variable. In order to place a sentiment, score our on preprocessed data we must come up with a way that reads our data most accurately. Textblob is an excellent package to preform sentiment analysis. After preprocessing our data, Textblob “goes along finding words and phrases it can assign polarity and subjectivity to, and it averages them all together for longer text.”(https://planspace.org/20150607-textblob\_sentiment/). Our result is that 1 is the most positive sentiment and –1 is the most negative sentiment. For the sake of having a target variable, we assume that the scores generated by Textblob are accurate.

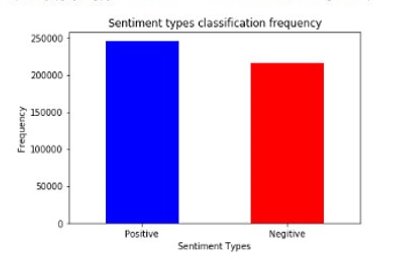


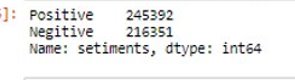
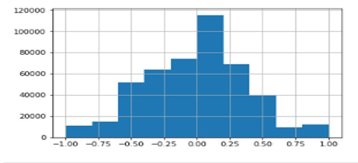
Now that there is a sentiment score the data is still unbalanced. As you can see most of our data lands on the neutral point of 0.



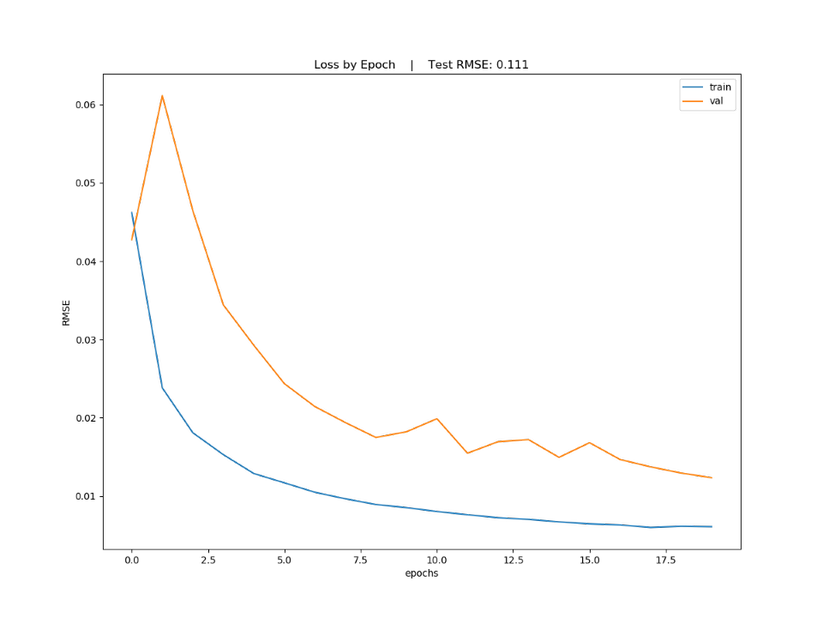


In order to balance our data, we must get rid of the neutral points since they serve no real purpose, because we are looking for >0< everything below and above 0 for positive and negative. This is our end result, binned into two buckets, and as a histogram of the scalar values.





After all the preprocessed data has been balanced, running the LSTM model was now possible. We ran the model to minimize RMSE. The LSTM allows for us to know what loss the model took while running the LSTM. As you can see the RMSE takes less Loss as the epoch increases. Up top we see the RMSE score for the hold out, test data. Inside the graph is our validation and training data results. The test RSME was 0.111, meaning that our model was off by an average of .011, considering that the range of the regression variable is 2, this seems good.



We also wanted to show our residual plot that allowed us to analyze the Textblob actual sentiment score data to the actual predicted sentiment, for each tweet.

