Fake Tweet Analysis

Authors:

Daniel Kim (014641497)
Simran Singh (010219339)
Omri Levia (014503619)

Source Code: https://github.com/brightdo/cmpe255-Fake-Tweet-Analysis

1. Introduction

1.1 Motivation

Social media platforms, especially Twitter, are special tools that can be used to elevate ideas, promote discourse, and can bring about rapid social change. Due to the mercurial nature of social media, discursive spaces like Twitter can present ideal conditions for bad actors to spread misinformation. Automated swarms of fake accounts can easily create millions of false tweets that pretend to come from real authors. Buntain et. al. remark that traditionally "journalistic gatekeepers" [4] would serve as the judges of truth, but with the rise of sophisticated technologies, the amount of information to label is far too high for a human to do alone; computational techniques are required to discern fact from fiction in tandem with human efforts. In light of this issue, this report details the implementation of several fake tweet classification models. Following Looijenga [3] we have implemented a subset of their SKLearn models, including multinomial Naive-Bayes, Random Forests, Decision Trees, Gaussian Naive-Bayes, Stochastic Gradient Descent, K-neighbors, and Linear SVM, although the linear SVM experienced timeouts. With recommendations from the literature, a convolutional neural network approach was also implemented with successful results.

1.2 Objectives

The objective of this report is to train several classification models to detect a fake tweet given the tweet's text; and in the case of the convolutional neural network, both approaches of using the tweet's text and authors will be applied.

1.3 Literature Review

Looijenga et. al. utilized several SKLearn classifiers for the task of Twitter hashtag classification [3]. Looijenga used a Bag of Words Representation model for preprocessing their data. They calculated word frequencies using a TfidfVectorizer (term frequency-inverse document frequency). Looijenga offers the formula for TF-IDF as follows:

$$\text{tf-idf}(t,d) = \text{tf}(t,d) \times \text{idf}(t)$$

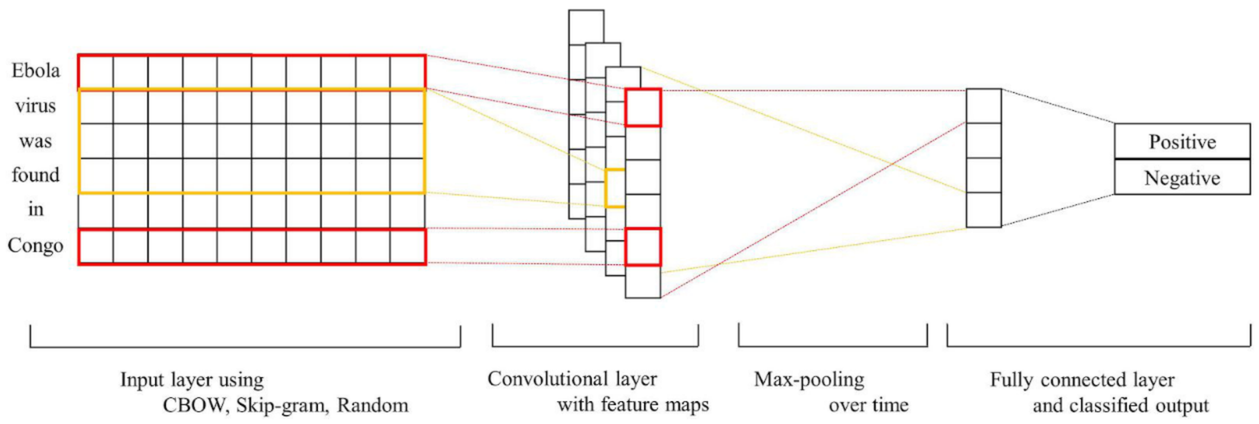
where *idf(t)* is:

$$\text{idf}(t) = log\frac{1+n_d}{1+\text{df}(d,t)} + 1.$$

Looijenga created eight classification SKLearn models: Linear SVM, Bernoulli Naive-Bayes, Multinomial Naive-Bayes, Gaussian Naive-Bayes, Decision Tree, Random Forest, Extra Trees, Stochastic Gradient Descent, and Random Forests. Out of these eight classifiers, the Linear SVM and Decision Trees performed the best with an F-score of 0.86 and 0.88 respectively. Lowest performing were Stochastic Gradient Descent and Bernoulli Naive-Bayes.

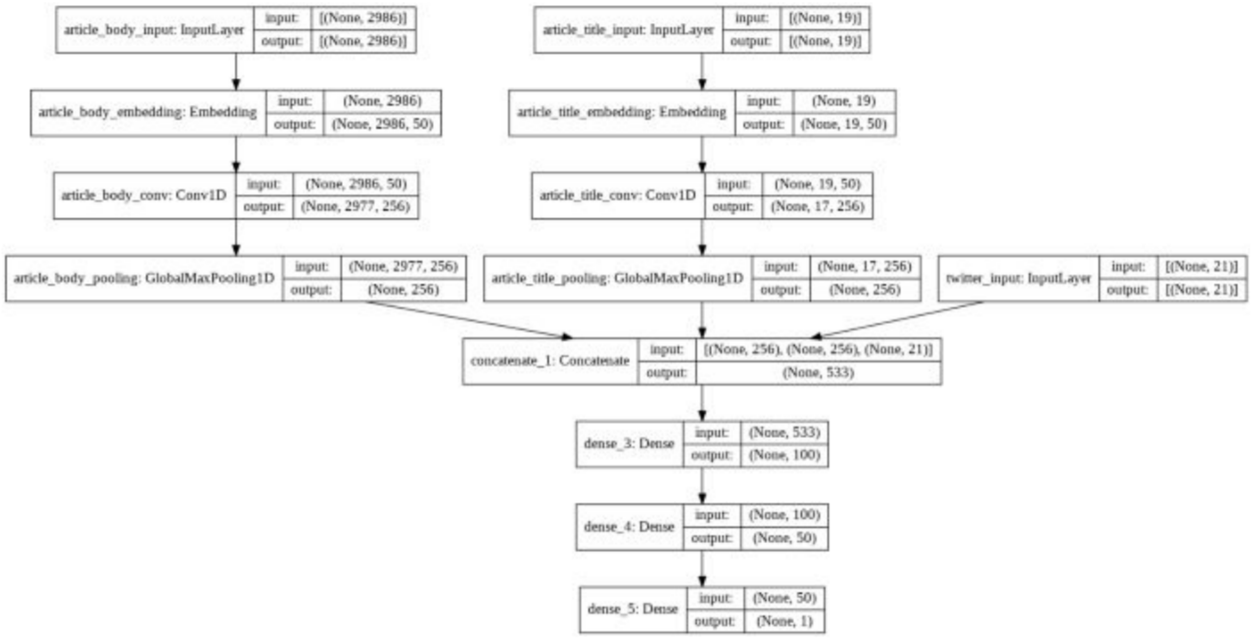
Ajao et. al. utilized a hybrid CNN approach to classifying fake news messages on Twitter with an 82% accuracy metric. Ajao et. al.'s approach involved "automatic identification of features [...] without prior knowledge of the subject domain or topic of discussion" [7]. They performed this by creating a hybrid model between deep learning LSTM (long-term short memory) and CNN models. They created three variants: LSTM recurrent neural network (RNN), which they used for sequence classification, LSTM with dropout regularization, and LSTM with CNN. Ajao et. al.'s dataset was 5800 tweets that involved five "rumor stories" [7]. Out of the three models, LSTM performed the best with an 82% accuracy score, following the LSTM with dropout and lastly LSTM with CNN. Ajao et. al. contend that "insufficient training samples" were responsible for lower accuracy scores in the latter two LSTM models.

Jang et. all utilized a Word2Vec approach [6]. According to Jang et. al., Word2Vec is a text vectorizer that leverages "similar meanings in a given context" for words with close distances. Jang et. al. also used a CNN model, noting their reasoning for the CNN power in text and image classification is due to the preservation of spatial information as features get mapped layer to layer. "Semantic similarity" facilitates the extraction of text characteristics in the convolutional layer and pooling layers. Jang et. al. performed classification with two embedding models, CBOW (continuous bag of words) and skip-gram, and then performed classification with CNN with each CBOW, skip-gram and word2vec. Text from their data was tokenized and given vector values by CBOW, skip-gram and random initialization algorithms [6]. Below is an image of their CNN architecture:


Ebola
virus
was
found
in
Congo
Positive
Negative
Input layer using CBOW, Skip-gram, Random
Convolutional layer with feature maps
Max-pooling over time
Fully connected layer and classified output

In their experimentation, CNN with CBOW performed the best with F1 score values ranging from 0.9341 to 0.9161.

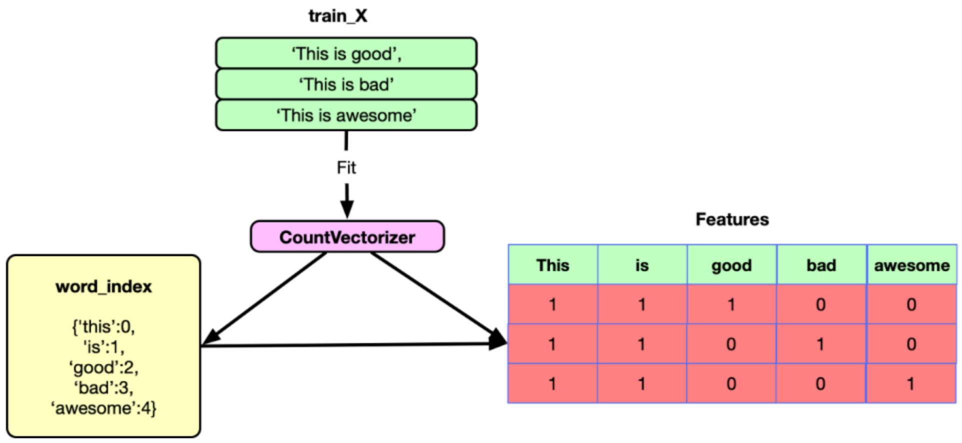
The next literature comes from Matthew Whitehead, author of the Medium article "How to Use Artificial Intelligence and Twitter to Detect Fake News." [2] Whitehead uses Oshikawa et. al's literature review to warrant the use of a CNN for text classification. Whitehead uses a dataset containing fake news articles embedded with Twitter posts. To make the dataset manageable and usable for the CNN, he tokenizes the text content using a Keras tokenizer, and then uses a text to sequences method provided by Keras to vectorize the text. Following, the text sequences are padded according to the mean length of the tweet content. Next Whitehead split the dataset into test and train, and used tensorflow's layers library to create the CNN layers. Whitehead's CNN architecture was constructed as follows:


article_body_input: InputLayer | input: [(None, 2986)] | output: [(None, 2986)]
article_body_embedding: Embedding | input: (None, 2986) | output: (None, 2986, 50)
article_body_conv: Conv1D | input: (None, 2986, 50) | output: (None, 2977, 256)
article_body_pooling: GlobalMaxPooling1D | input: (None, 2977, 256) | output: (None, 256)
article_title_input: InputLayer | input: [(None, 19)] | output: [(None, 19)]
article_title_embedding: Embedding | input: (None, 19) | output: (None, 19, 50)
article_title_conv: Conv1D | input: (None, 19, 50) | output: (None, 17, 256)
article_title_pooling: GlobalMaxPooling1D | input: (None, 17, 256) | output: (None, 256)
twitter_input: InputLayer | input: [(None, 21)] | output: [(None, 21)]
concatenate_1: Concatenate | input: [(None, 256), (None, 256), (None, 21)] | output: (None, 533)
dense_3: Dense | input: (None, 533) | output: (None, 100)
dense_4: Dense | input: (None, 100) | output: (None, 50)
dense_5: Dense | input: (None, 50) | output: (None, 1)

Using this architecture Whitehead achieved an F1-score of 0.80.

Another article we reviewed was by Mohamed Afham from Towards Data Science which was titled "Twitter Sentiment Analysis using NLTK, Python". Afham uses Python's Natural

Language Toolkit (NLTK) to perform some sentiment analysis on tweets. Along with NLTK, he utilized the TextBlob library to do important preprocessing of his dataset. Using SciKit's guide on working with text data he chose to adopt the Multinational Naïve-Bayes algorithm along with a count vectorizer. The count vectorizer works as follows:



This	is	good	bad	awesome
1	1	1	0	0
1	1	0	1	0
1	1	0	0	1

The words themselves are identified using the feature selection from the TextBlob and further normalized by lemmatization [10].

2. Design & Implementation details

2.1 Algorithms used

As mentioned in the literature review, the algorithms that this report explored proved to be the most effective in various text classification problems. In this report, a CNN approach like that of [2] was implemented with results that exceeded expectation. In conjunction with this, two Naive-Bayes implementations were tried, as well as other contenders like Linear SVM, Decision Tree Classifier, Stochastic Gradient Descent, K-neighbors, Logistic Regression, and Random Forests. The testing of all of these algorithms will serve as a comparison with the literature, as well as make for an experiment to see which algorithms are best suited for text classification. Finally, these algorithms will be compared to each other to identify the best performer relatively.

2.2 Technologies and Tools

For the implementation of the CNN, an interactive python environment Google Colab was used. Google Colab is useful in the ease of configuration of TensorFlow. Google Colab also offers the benefit of GPU acceleration for quicker computation, all while in the browser. TensorFlow was used in the creation of the CNN layers, as well as for the tokenization of the text. For the other models explored, the SKLearn classifiers were used, following Looijenga et. al.

2.3 Architecture Related Decisions

In the CNN model, two approaches were used. One approach was to use only the tweet text as input, and in the other approach the tweet author and the text was considered. As such, two CNN models were created. In the text only approach, a sequential layering model is used, and in the author and text approach, both author and text are used as inputs into separate layers and concatenated in a later layer stage. The sequential approach features an embedding layer, followed by a 1D convolutional layer, followed by a pooling layer, then a flattening layer, then two dense layers. The non-sequential model puts the author input into an embedding layer, a convolutional layer, then a pooling layer. It is then concatenated with the tweet input, and placed into two dense layers.

3. Experiments/Proof of concept evaluation

3.1 Dataset and Preprocessing

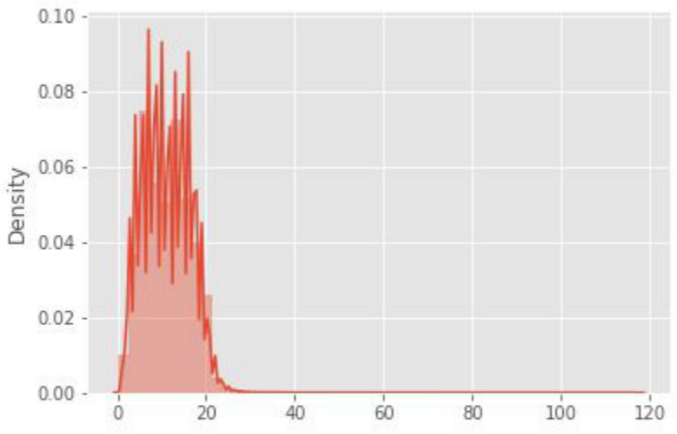
The dataset used for this report is the conglomeration of two principal sets taken from Kaggle. The first is 3 million troll tweets [1] from FiveThirtyEight. These tweets are the product of a troll factory. The tweets were obtained using a tool called Social Studio, created by Salesforce and authorized for use for Clemson's Social Media Listening Center [1]. The 3 million tweets were authored by just under 3000 unique accounts. The dataset contains several attributes in addition to the tweet text itself, like author, region, language, publish date, followers, account category, and so on. In this report, only the author and content were retained, since no other attribute was necessary for the experimentation. In order to train the classifiers, real tweets needed to be added to the set. For this, 1.6 million tweets from a sentiments analysis set was used [9]. These tweets have been gathered from veritable accounts and are labeled as real tweets. The fact that they were gathered from a sentiment analysis set is not of significance, all that matters for the experimentation is that they are real tweets. Like the fake tweet set, only the author and text were retained in the set.

Following the retrieval of the sets, half of the fake tweets were sampled, to create about an even amount of real to fake tweets, for a total of just over 3 million. The fake tweets were labeled with class '1', and the real tweets were labeled as class '0'. The resampling of the fake tweets was performed using the pandas sample method, with replacement set to False. This way, about an even amount of accounts were represented and the analysis will not have been skewed. The two sets were then combined to create the whole dataset.

The next task was to clean the text content for stop words, punctuation and symbols. Before this, all non-english rows of the set were removed. The NLTK corpus library stopwords was used to omit stopwords. After stop words were removed, all symbols and punctuation were as well from the text. This way each text entry in each row only contains an alphabetic string.

3.2 CNN Methodology and Results

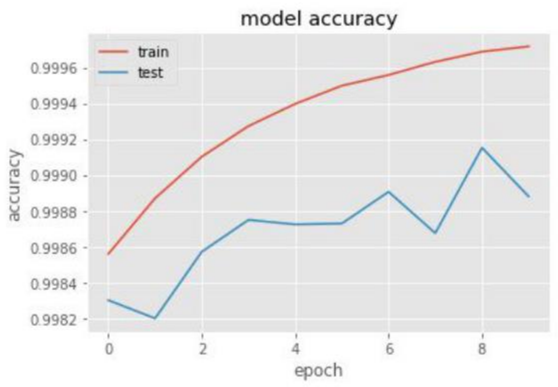
As discussed earlier, two CNN implementations were applied. The first using a sequential model with only text as input and the second using both author name and text as input. Both methods underwent the same preprocessing methods as described above. After preprocessing, the text content that comprises the tweets was tokenized using a Keras text tokenizer. For both implementations this was done, for text and text and author alike. Since input to the model must be uniform, some of the tokenized text array needed to be padded. In order to find out the length of text content to use, the mean text array length was found as well as the standard deviation. For the text itself, the max length of the text array was set to be the mean plus twice the standard deviation. Below is a graph displaying the text length distribution:

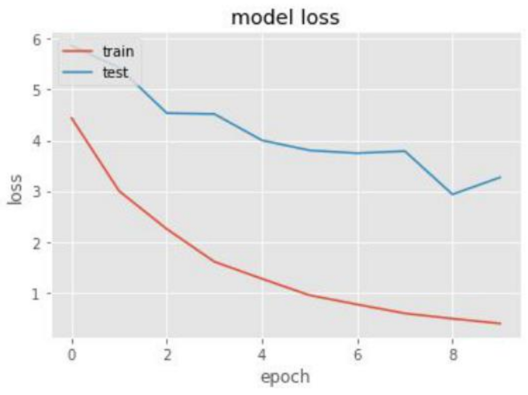


We can see in the figure that many of the text sequences fall below 20 words. The mean was found to be 11.1, and the standard deviation 5, so the max length for text was set to be 21.

This was similarly done for the implementation containing the author, where the mean author length was 1.1, and standard deviation 0.35. The max author length was taken to be 3, however, since this captures 98% of all cases, whereas a length of 2 only captures about 95%. Using the max length values, the text sequences and in the case of the author as well, were padded. Next came the creation of the CNN models. In the sequential and non-sequential model, the CNN layers were created as is described in section 2.3.

Test and training sets were created using SKLearn’s train_test_split, using default test size of 25% of the dataset. The sequential model was trained over 29 epochs, and only achieved an accuracy score 0.502. Either the layering scheme was invalid, or for this application, text alone was insufficient in creating viable discrimination. In the non-sequential model, with author input as well, the model was trained over 10 epochs, with a final accuracy of 0.9989. Surely this jump was due to the input of the author names, which correlate heavily with the false tweets. Below can be seen the evolution of accuracy and loss in the non-sequential model over the 10 epochs:





3.3 SKLearn Algorithm Methodology and Results

As was performed with the CNN implementation, the dataset for the SKLearn experiments was removed of stopwords and punctuation. Following the analysis of Looijenga, SKLearn's TF-IDF transformer from the feature extraction library was used for tokenizing the text. Due to the large number of dimensions--almost 2 million--truncated SVD was used to decrease the number of dimensions. The number of components retained was 16 for the dataset, capturing the majority of the variance in the newly formed attributes. Below is a table of results for the SKLearn algorithms tested:

Classifier	**Accuracy Score**
Logistic Regression	0.7857
Stochastic Gradient Descent	0.7738
K-neighbors, n_neighbors=4	0.8497
Decision Tree	0.8169
Random Forest	0.8851
Linear SVM	Execution timeout

No SVD

Classifier	**Accuracy Score**
Logistic Regression (max_iter =10000)	0.93
Stochastic Gradient Descent (state=42,max_iter=5, tol=None	0.88

K-neighbors, n_neighbors=4	Execution timeout
Decision Tree	0.63
Random Forest	Execution timeout
Linear SVM	Execution timeout

3.4 Multinomial Naïve-Bayes Methodology and Results

In this approach, the “tweets.csv” file is inputted into a Jupyter notebook running Python 3. Pandas is used to read in the first 2 million rows as the training set and the first 10,000 rows as the test set. The train set keeps it’s “class” column along with the tweet “content” column and the test set will use only the “content” column for evaluation purposes. Using NLTK, a text processing function is created with TextBlob being used to split each tweet into separate words and then remove punctuations and english stopwords like “and, the, or, etc”. Finally, the words are lemmatized (normalized). This text processing function is applied to both the train and test data and inserted to the model pipeline. The pipeline is as follows: a count vectorizer is used to assign an integer to each word, these integers are converted to TF-IDF scores, and finally SKLearn’s Multinomial Naïve-Bayes is used to classify these TF-IDF scores. Finally, the evaluation metrics are exported.

As seen from the source code. Our dataset got an average accuracy of 96% using the methodology described above. The confusion matrix is as follows:

```
[[318561 14577]
 [ 1316 65546]]
```

We did see that the selection of the subset used as the training data set from the full tweets.csv file does change the precision of our “fake” tweets classification. The “real” tweets classification has 100% precision while the “fake” has 82%. However we do see that the support count for the “real” tweets is almost 5 times more than the “fake” tweets. All in all, the results show that the Multinomial Naïve-Bayes solution to classifying fake tweets was a good choice however we did see a long processing time due to the sheer size of our dataset. For future experiments, an algorithm that works more efficiently time wise should be considered with some obvious tradeoffs in accuracy.

4. Discussion & Conclusions

We started with two csv files each with only 3 million troll_tweets and 1.6 million real_tweets that has different number and kinds of columns, our group had to decide on a format to combine these two csv files into one with the correct kind of column to be used for our classifiers. In result we decided to keep all the real_tweets and shorten the troll_tweets into 1.6 million rows to match the number of rows of the real_tweets to make a 1:1 relationship. We also decided to keep only the text content column of the initial files because we were looking into text classifying algorithms for this task. Upon creating our base csv data file to be used, we realized that the content of each tweet was messy, so we had to do some research and make a decision on which text pre processing techniques to use. One of the main and biggest questions our group had to answer was which algorithms to use to create our model. For this, we all read the research paper by Looijenga.

There were some difficulties faced during the creation of our classifier model. Our very first CNN model was showing a very poor accuracy of 0.502 which is nothing better than a random guess algorithm considering the result being only 0 (real tweet) or 1 (fake tweet). Also some of our earlier tweet text processing methods had some difficulty diving a sentence into a list of words as intended. Sometimes a word would be put in an index one bigger than the rest due to an uncatched space or non alphabet character. This additional index would lead to some problems such as exporting the result database to a new csv or a loop through each row of the column failing due to an empty value. There was some difficulty due to the large dataset itself. The 3.2 million total tweet and its 11.1 mean text sequence made it make many of our models take a significant amount of time. Even after applying the SVD, the execution time was very long. Even though the theoretical results have pointed towards a smaller accuracy than the 96% we achieved with the Multinomial Naïve-Bayes model, we would have liked to see the result from the model created using Google's pretrained BERT and ELMo models. These models did show some new innovation that would have been a valuable learning experience but like the SKLearn's KNN model without SVD, the execution time was long due to the sheer size of our dataset. In the future we would like to try these approaches on CUDA or IBM Cloud to shorten the execution time to a reasonable range.

Overall the result we have seen on our models is satisfactory. The CNN model with not just the text content, but also the author had a great result of 99.89%. All the fake tweets come from only 3000 authors, however, and using the author in the CNN model might have made it overfitted, so this is something to consider when evaluating the accuracy score. Another thing we are satisfied with about our result is the accuracy score from the SKLearn algorithms. Although these didn't get a high accuracy as the CNN or the multinomial Naïve-Bayes algorithms, their results compiled to the ones that Looijenga's proposal of Stochastic Gradient Descent performing the worst and the Decision Tree performing comparatively better. This tells us that our pre-processing algorithm which accounts for a significant portion for deciding a text classifier's performance worked very well.While some of the group's plan went well and returned satisfactory results, some plans didn't go so well.

One of our group's plans was to create a Naïve-Bayes algorithm from scratch and compare its results to the one from the SKLearn library. We would see each word in the tweet and multiply the probability that the tweet would be a troll_tweet if that word is present and compare that to the resulting multiplied probability of the tweet being a real_tweet with each of those words present. The plan didn’t work so well with the complication of each step and the large number of our data set. Although we were not able to create a satisfactory end result, we were able to learn a lot about each of the steps for the Naïve-Bayes algorithm for text classification.

Our research has shown that CNN model has the best accuracy of 99.89% when author name and text is both used as input, however this suspiciously high number of accuracy is something we could look into and see if the author did make the model overfitted. Training the same algorithm with a different data set with a more variety of users would show us a more accurate indication of this model. Considering this inaccurate accuracy representation from using the author of the tweet, The Multinomial Naïve-Bayes algorithm still showed a good accuracy of 96% while using only the text content of the data set. However this high accuracy is also still a value much higher compared to the other algorithms from the SKLearn library. This is something our group would like to look into deeper in the future.

5. Project Plan / Task Distribution

In this project the tasks were distributed as follows: curate the dataset as a group, preprocess the data as a group, Omri would then handle the CNN implementations, and Daniel and Simran would handle the SKLearn algorithm implementations. As such, as a group we looked for the 3.2 million troll_tweets and 1.6 million real_tweets from Kaggle and debated about how to partition the two csv files to create one big csv file all of us would use together. And as a group we created the python code for the partitioning and appending of 1.6 million tweets from both csv files into the one tweets.csv. Only the data meaningful to our implementation of classifiers was extracted from both files in order to save processing time and also avoid skewing our results.

Initially we each individually worked with the curated dataset to work with each assigned algorithm. Later when we met again with individual codes and parsing methods, we talked to each other about the problem or strength of each of our partitioning methods and combined them together to create the one we use as a group.

After curating the dataset and reading through the research paper by Looijenga, our group decided to each deal with an algorithm. Omri implemented the CNN, Simran implemented the multinomial Naïve-Bayes algorithm through SKLearn, and Daniel initially implemented the Naïve-Bayes algorithm from scratch. After some time, the group decided that the implementation of Naïve-Bayes from scratch was not a good idea and Daniel worked on Stochastic Gradient Descent. Afterwards, Daniel and Simran worked on the other SKLearn algorithms: (Logistic Regression, K-neighbors, Decision Tree, and Random Forest). Everyone did the task they were assigned.

References

1. https://www.kaggle.com/vikasg/russian-troll-tweets
2. https://medium.com/better-programming/how-to-use-artificial-intelligence-and-twitter-to-detect-f ake-news-a-python-tutorial-75a4132acf7f
3. http://essay.utwente.nl/77385/1/Looijenga_BA_EEMCS.pdf
4. https://arxiv.org/pdf/1705.01613.pdf
5. https://www.kaggle.com/fivethirtyeight/russian-troll-tweets
6. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0220976
7. https://dl.acm.org/doi/abs/10.1145/3217804.3217917
8. https://arxiv.org/abs/1811.00770
9. https://www.kaggle.com/kazanova/sentiment140
10. https://towardsdatascience.com/twitter-sentiment-analysis-classification-using-nltk-python-fa912578614c