CPS 803 Report
Fake News Detection

Christian Wang
Computer Science
Ryerson University
Toronto, Ontario

Da Hee Kim
Computer Science
Ryerson University
Toronto, Ontario

I. Introduction

A. Problem

We began with the idea to train a model that can detect whether a given news statement is a fake machine-generated statement. The problem we tackled was to detect if a given news statement is fake, not specifically machine generated. We wanted to take text from news and identify it as fake or real.

B. Significance and Challenge

The growth of social media and world-wide communications are the everyday individual's cause for concern. It is becoming increasingly difficult for people to discern the correctness of the media. Furthermore, with the emergence of "fake news" generating bots and unhinged popular figures, it is ever-more important to have a reliable way to filter out inaccurate media.

The challenge of this problem is that news covers a broad variety of subjects such as politics, business, sports etc. To add on to that, the definition of fake news is not as clear cut as, for example, the definition of what a cat is. For our purposes, we used preexisting datasets with labels assigned to samples. However, in the future, we might process our own samples and give them a false or true label according to a certain standard that classifies fake news.

C. Prior Work

One example of prior work is an article by George [3]. The author [3] also uses natural language processing to detect fake news using Dataset 1 [1]. For feature extraction, she uses a CountVectorizer and TfidfTransformer [3]. And for the models, she uses a Naive Bayes, SVM, and Passive Aggressive Classifier [3]. In conclusion, she found that the SVM and Passive Aggressive Classifier performed better than the Naive Bayes model [3].

D. Overview of Report

In our report, we will be detailing the datasets we used and the feature extraction method we used on our datasets. We will also discuss the models we used to address the problem of fake news. And most importantly, we will examine and present the results of our experiments.

II. Datasets

Two datasets were used:

1. Clement Fake and real news dataset [1]
2. Comp Fake News dataset [2]

The distribution of classes for real news and fake news of the datasets can be seen in figure 1.

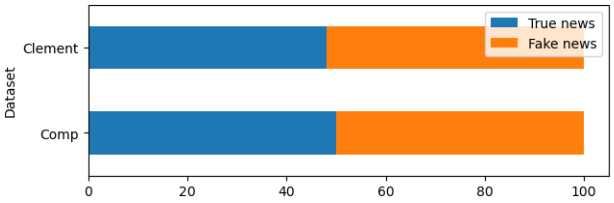

Figure 1 Distribution of Classes in Datasets

For Dataset 1, the fake and true news were divided into 2 files, so we combined them and added a label column which indicated 0 for fake news and 1 for real news. In total, Dataset 1 has 44,898 samples.

For Dataset 2, we used the train.csv file from the site. In total, dataset 2 has 20,800 samples.

III. Methods and Models

The overall methods of Natural Language Processing can be described in the following flow chart, refer to figure 2.

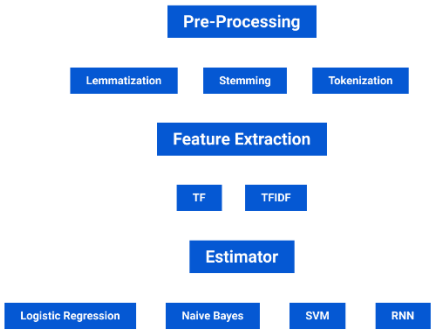

Figure 2 Text Classification Model Flow Chart

Now, we move to explain each specific method.

A. Pre-Processing and Text Normalization

When splitting the datasets, we first mixed the samples randomly. Then we split the data into training, validation, and test files by 70%, 15%, and 15% respectively.

Pre-processing was done through the Count Vectorizer and TfidfVectorizer from the scikit-learn package. Text was converted to lowercase and stop words were removed.

B. Feature Extraction

We used 2 different feature extraction methods for the models, to effectively derive features from the training examples and build a vocabulary of words. The methods are similar but differ slightly when it comes to assigning feature weights.

1. Term Frequency (TF): This is the most common method and is simply the number of occurrences of the given term in the given example. All terms are treated equally. The mathematical representation is as follows:
    a. TF(term) = (number of times term appears in a document) / (total number of terms in the document)
2. Term Frequency Inverse Document Frequency (TFIDF): Unlike TF, TFIDF does not treat terms equally, TFIDF attempts to score terms by importance. This is done by multiplying the Inverse Document Frequency, IDF with the Term Frequency, TF to determine the feature weight. The mathematical representation is as follows:
    a. TF(term) = (number of times term appears in a document) / (total number of terms in the document)
    b. IDF(term) = log((total number of documents) / (number of documents with term in it)
    c. TFIDF(term) = TF(term) * IDF(term)

Additionally, certain terms or stop words were left out of the vocabulary. This included common words like “a” or “the” which would appear indiscriminately frequently in both classes. We also used n-gram to determine size of terms. Unigram terms are made up of single terms or words, while Bigram terms are made up of two terms or words. Employing Bigram in the term vocabulary can add a layer of sequential logic to the model.

C. Specific Learning Algorithms

For the Fake News Classifier, we tested multiple different learning algorithms to see which would perform better. Initially we tested classical learning algorithms like Naive Bayes [C1] and Logistic Regression [C2] and later moved on to Support Vector Machines [C3] and Recurrent Neural Networks [C4].

C1. Naive Bayes

Naive Bayes classification is a simple probabilistic classifier that works by applying Bayes Theorem. Using an assumption conditional independence for every feature This approach calculates the probability that a feature is indicative of a class which can then be used to calculate the probability that a sequence of features, a sentence or statement belongs to a class.

C2. Logistic Regression

Logistic regression classification belongs to the large general family of linear model classifiers. This algorithm attempts to learn the optimal decision boundary that separates features from classes.

C3. Support Vector Classifier (SVM)

The Support Vector Classifier is to determine a hyperplane between classes. This is done using the kernel trick, a method that maps input data to higher dimensions according to a specified kernel function. For this model, we utilized the linear kernel function, which determines linear separations of features between classes.

C4. Recurrent Neural Network (RNN)

Recurrent Neural Networks (RNN) are frequently used for modelling time series or natural language data. The unique perk of the RNN is the addition of a Long Short-term Memory Layer (LSTM) which connects the output of the previous neuron with the next, thus creating a recurrent messaging system that acts as a sort of pseudo memory. This is especially useful for natural language applications as the RNN can effectively learn and predict the next word in a sentence, given the previous words.

The architecture of our base RNN consisted of 3 layers, the input/embedding layer, LSTM layer, a ReLU activation layer and the output layer, refer to figure 3.

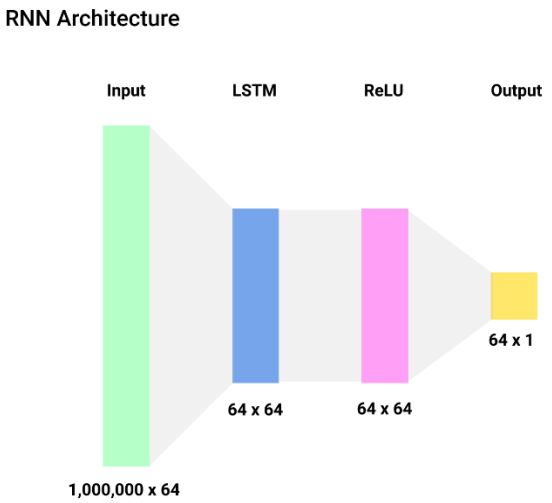

Figure 3: RNN Architecture

D. Performance Metrics and Post-Processing

All models were scored against the test dataset for accuracy. Training and validation scores were determined using 5-fold cross validation, see figure 4.

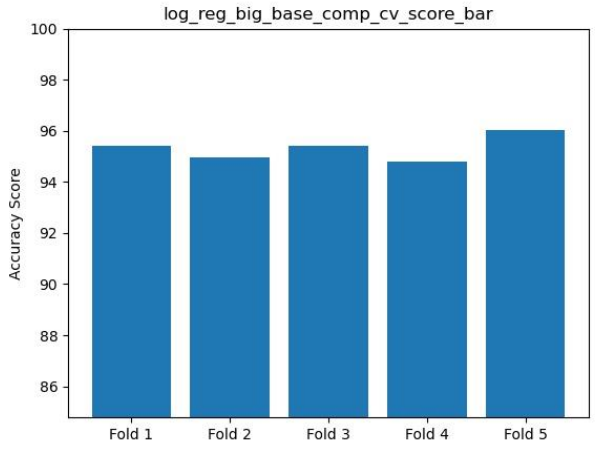

Figure 4: K-Fold scores for Log Reg on Comp Dataset

The CV scores were then averaged to give a mean CV score and distribution. Additionally, models were applied to a confusion matrix to provide more information on test accuracy, see figure 5.

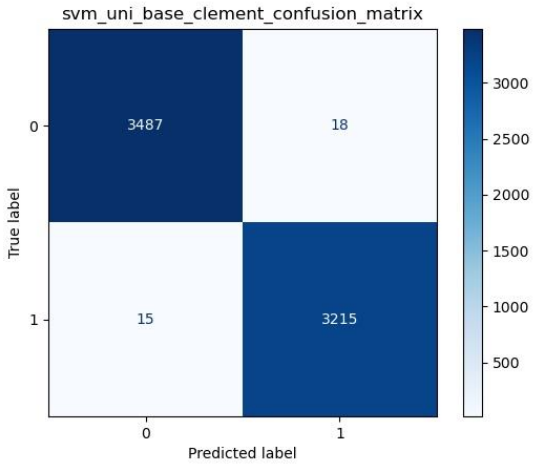

Figure 5: Confusion Matrix of SVM on Clement Dataset

After analyzing the learning curve of models, we determined that simply using default parameters for each of these models wasn't enough, see figure 6. We found that the process of adjusting hyper-parameters and feature extraction methods was continuous to find the best fit for each model. Using the metrics gathered, we were able to diagnose model problems like overfitting and underfitting. Testing different hyperparameters and extraction methods allowed us to come up with generalized results.

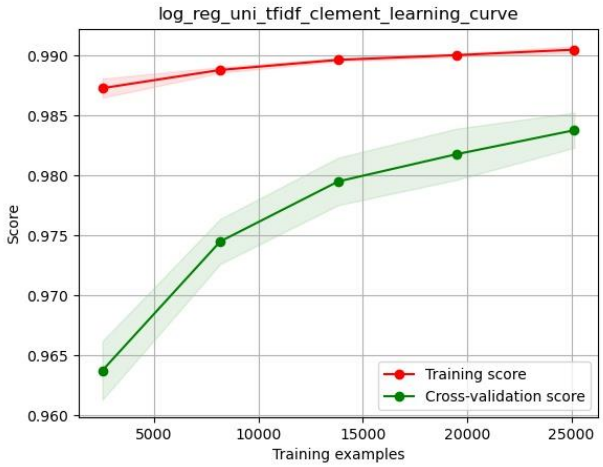

Figure 6: Learning curve of Log Reg on Clement Dataset

IV. Results and Discussion

We were able to classify news statements as real or fake with relative accuracy with the models we employed. On both datasets, all models were able to classify with greater than 85% accuracy, with top models, RNN performing greater than 98% and 99% on the comp and clement datasets, respectively. Furthermore, we were able to determine the top indicative words for classifying fake news, which can be seen from a word cloud in figure 7.

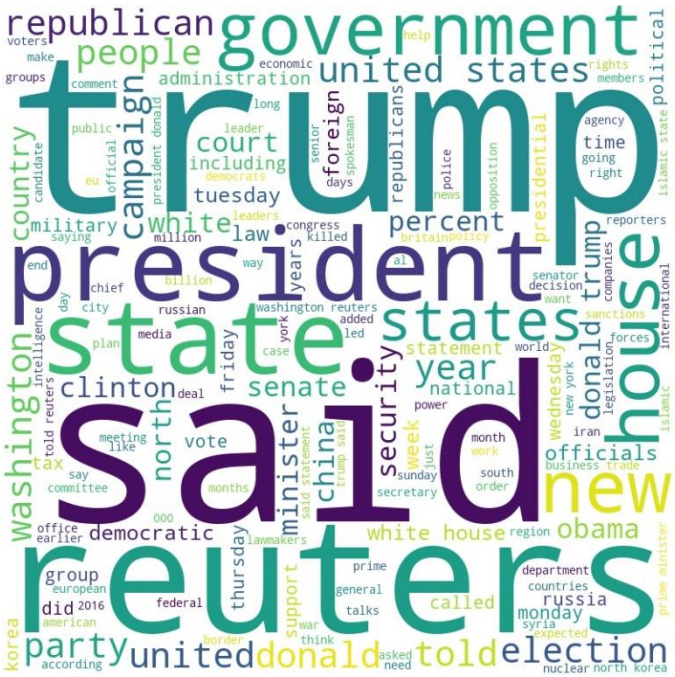

Figure 7: Word Cloud of Top Indicative Words Naive Bayes on Clement Dataset

A. Model Comparisons

Overall, the RNN model performed best on both datasets. All the RNN models scored higher than the other models. The ranking of the other models differed based on the n-gram and the feature extraction method. In general, Logistic Regression models and SVM models performed similarly and Naïve Bayes models usually underperformed compared to the other models, see figures 8 and 9.

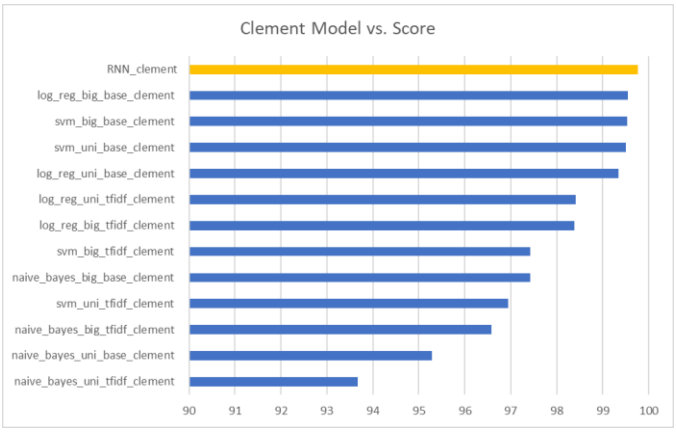

Figure 8: Model Score Comparison on Clement Dataset

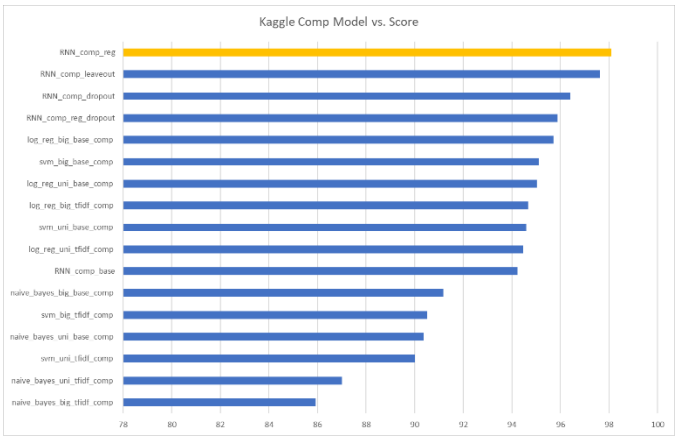

Figure 9: Model Score Comparison on Comp Dataset

We started with classical machine learning algorithms like Logistic Regression and Naive Bayes due to the simple complexity and faster training times. We focused on optimizing the models but found that they seemed to reach a performance limit. The addition of the RNN model greatly improved performance, at the cost of training and computational time. We suspect that this was the case due to the nature of the RNN algorithm compared to the other models. Since the RNN algorithm uses a learning/prediction system based off sequences of words while the other methods rely on learning/prediction from individual words.

B. Term Frequency vs TF-IDF

As for feature extraction methods, we found that across all models, Term Frequency outperformed TF-IDF, see figure 10. For each model with their TF vs TF-IDF variations, the TF counterparts always outperformed the TF-IDF.

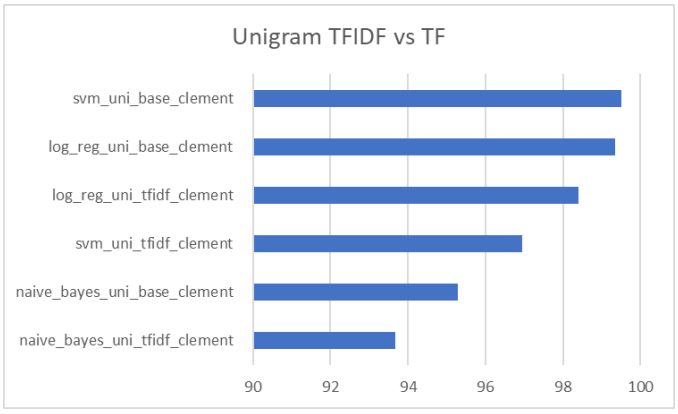

Figure 10: TF vs TF-IDF on Clement Dataset

With NLP problems, it seems like the domain of data strongly determines the results. Since TF-IDF flags terms that appear in many examples, we suspect the TF-IDF scheme was incorrectly weighing important features. Probably, certain words that were appearing multiple times in different documents which were indicative features were being penalized by the TF-IDF algorithm, leading to lower accuracy results.

C. Unigram vs. Bigram

For term-frequency, models using bigram outperformed their counterpart models using unigram, see figure 11. Since bigram adds a layer of sequential logic, it would make sense that models employing the method perform better.

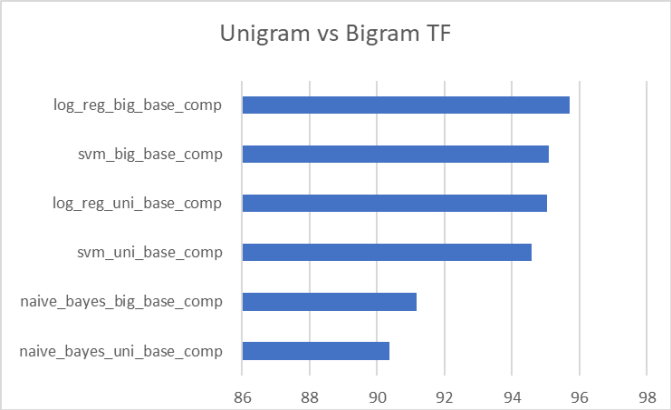

Figure 11: Unigram vs Bigram TF in Comp Dataset

For TFIDF, results were more inconclusive. For the most part, bigram still outperformed unigram. On the clement dataset, this was the case except for Logistic Regression, see figure 12. However, both Logistic Regression models had very close accuracies, a 0.03% difference.

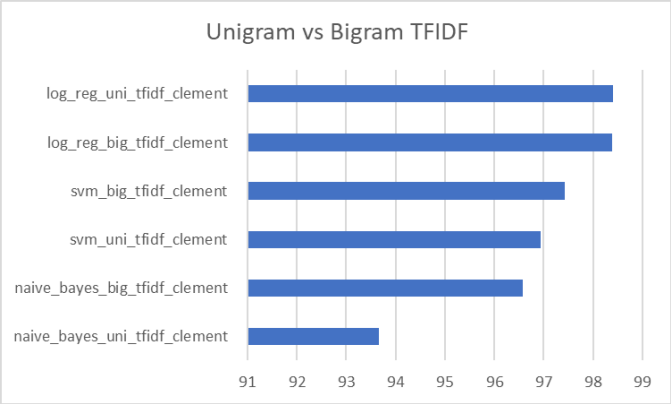

Figure 12: Unigram vs Bigram TFIDF in Clement Dataset

Similarly, on the comp dataset, the naïve bayes model using unigram outperformed the bigram variant, with a difference of 1.09%, see figure 13. These different results lead us to conclude that TFIDF in combination with changes in n-gram lead to slightly ambiguous scores. We suspect this could be the case because of the lack of training examples for the bigram terms; since the occurrence of unique bigram terms would be far less than the occurrence of unique unigram terms, it could be the lack of enough training data causes these discrepancies in results.

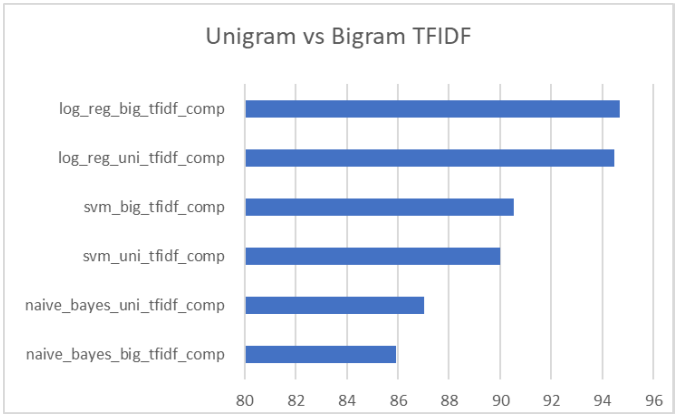

Figure 13: Unigram vs Bigram TFIDF on Comp Dataset

C. RNN Optimization

By default, the RNN models outperformed the other models on both datasets. For the Clement dataset, the RNN model had an accuracy score of 99.77%. However, for the comp dataset we found after analyzing the learning curve, the RNN seemed to suffer from overfitting, see figure 14.

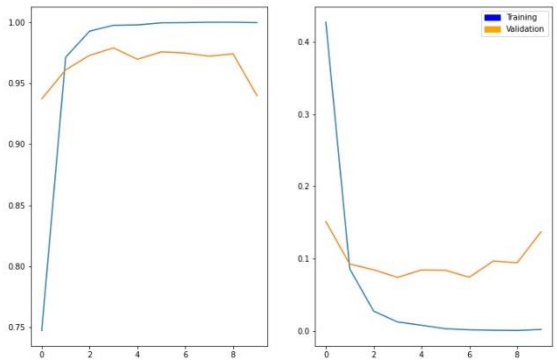

Figure 14: RNN Baseline Learning Curve on Comp Dataset

As visible on figure 14, the plot on the left denotes the accuracy over epochs and the plot on the right denotes the loss over epochs. It is clear to see that the model accuracy dips and the loss spikes after epoch 8. So, we employed a few different methods to attempt to solve the overfitting.

First, we used the simple leave-out method to stop training before the point of overfitting. The leave-out model was trained until just before epoch 7 and seemed to perform better than the baseline model. The model wasn't overfitting, but the accuracy could have still been improved. Leave-out while effective for the specific dataset isn't a viable option for creating a good, generalized model.

Next, we tried implementing drop-out layers in between the layers of the RNN. This also improved the accuracy of the model but was still underperforming when compared to the leave-out method.

Finally, we tested out adding L2 regularization to the RNN. This provided the largest performance boost, outperforming

all the models including the leave-out. The model was no longer overfitting the data and had a much higher accuracy score, see figure 15.

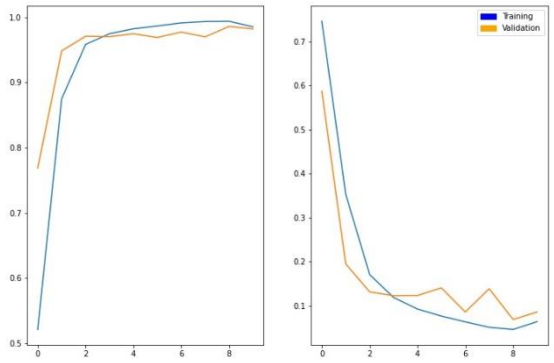

Figure 15: RNN Regularization Learning Curve on Comp Dataset

The total performance increases of all RNN models can be seen in figure 16.

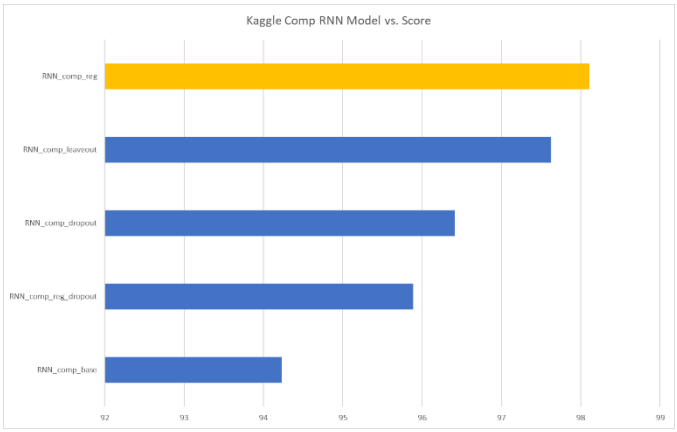

Figure 16: RNN model performances on Comp Dataset

V. Implementation and Code

The code and implementation for this project is publicly available on GitHub (https://github.com/Squishy123/CPS803-Project).

For our Logistic Regression model, Naive Bayes model, and SVM model, we used the Scikit-learn package. Models and variants were created using Scikit's pipeline methods. All models were fit, run and tested on the clement and comp datasets. TF and TFIDF were implemented using Scikit's CountVectorizer and TfidfVectorizer, respectively. N-gram was passed in as one of the optional parameters for these methods. For the Logistic Regression model GridSearchCV was used to determine the most optimized hyperparameters, but the resultant estimator contained the default parameters.

For our RNN, we used TensorFlow and Keras. The input data and vocabulary were built using the experimental TensorFlow TextVectorization encoder. The architecture of the model was built using the Keras models provided by the TensorFlow API. The model was trained on a GCP AI notebook instance with GPU support.

VI. References

[1] "Fake News Classification Project" https://github.com/Squishy123/CPS803-Project

[2] "Fake and real news dataset." https://kaggle.com/clmentbisaillon/fake-and-real-news-dataset (accessed Oct. 1, 2020).

[3] "Fake News." https://kaggle.com/c/fake-news (accessed Dec. 11, 2020).

[4] J. A. George, "Fake News Detection using NLP techniques," Medium, Dec. 04, 2020. https://medium.com/analytics-vidhya/fake-news-detection-using-nlp-techniques-c2dc4be05f99 (accessed Dec. 12, 2020).