Identifying Patterns and Trends in Fake News Articles

Ishaan Singh
Electrical and Computer Engineering
University of Florida
Gainesville, Florida
ishaans1999@gmail.com

***Abstract*—Fake news detection is an important new research area that can help dismantle misinformation in a contentious political landscape. This project trained and compared the performance of various natural language classifiers to be able to detect fake news articles from their headlines and body text. Analysis on n-gram use in feature extraction and document frequencies per term was conducted, as well as sentiment analysis on the data. It was found that increasing minimum document frequency past 20 or 30 documents has an adverse effect on the accuracy of training, and that uni-grams were most essential to training of the models. As expected, the linear models performed much better than the nonlinear models, with the linear SVM model performing best. It was also found that the models tend to predict false negatives more than false positive. In the future it would be beneficial to add more data, add sentiment based analysis, and create methods of deep syntax extraction.**

Keywords—fake news, machine learning, misinformation, text classification

I. Introduction

Fake news detection is an important new research area that can help dismantle misinformation in a contentious political landscape. This project aims to train a natural language classifier to classify news articles as either fake news or not, by analyzing the words in the news headline and the article's body text, and training and comparing various machine learning models and feature extraction parameters. Fake news was defined as news articles that are intentionally written to mislead the viewer, contain conspiracy theories or unfounded rumors, or contain other misinformation. For this project, the FakeNewsNet dataset was used, which contains English news articles that were fact checked from the fact-checking websites politifact and gossipcop [1]. This project only focused on the political news from politifact. Using Python with the nltk and sklearn packages, multiple natural language processors were trained to test which models perform better in this application [2] [3]. There was also an effort to categorize which range of n-grams work best for training a natural language classifier, as well as further tuning of the feature extraction model, detailed in Sections II and IV.

II. Description

A. Downloading dataset

The dataset was downloaded as per the instructions on the FakeNewsNet Github repository [4]. As mentioned previously, only the politifact data was used. This data is not the most recent data collected, and there were some issues with the article scraping, specifically with articles being deleted.

B. Pre-Processing Data

To start with pre-processing, each article's title and body text were combined into one string object and each word in the string was stemmed. Stemming is the process of returning words to their base/root form. For example "turning" or "turned" would both go to "turn." The Snowball stemmer, initially developed by Martin Porter, was used for this project, specifically the English (Porter2) algorithm [5].

The rest of the pre-processing was done with the TF-IDF feature extraction vectorizer from sci-kit learn [6]. It removes any accents from characters, removes stop words from the documents, and tokenizes the text before feature extraction. Stop words are extremely common words that occur in the English language that are removed from the text; the sklearn function uses their own built in stop word set. Finally, tokenization of text is splitting up the text into a list of words, removing any text that is not matched by a regexp pattern, detailed in Section IV.

The sentiments of the real and fake news sets were analyzed using the NLTK VADER sentiment analyzer, which generates a compound score between -1 and 1 of the sentiment of text.

C. Feature Extraction

The feature extraction model used was TF-IDF, or term frequency-inverse document frequency, which assigns weight terms to words in documents with the following equation:

$$TFIDF = TF * IDF \tag{1}$$

$$TF(t,d) = \frac{f_{t,d}}{\sum f_{t\prime,d}} \tag{2}$$

$$IDF(t,D) = log\frac{N}{\{d \in D: t \in d\}} \tag{3}$$

Where *TF* is the term frequency and *IDF* is the inverse document frequency. In the TF equation, *f* is the raw count of a term in document *d*, and the denominator is the total number of terms in the document. In the IDF equation *N* is the number of documents in the corpus, and the denominator is the number of documents containing term *t*. This essentially weighs how common the term is per document against how common the term is across all documents, which will weigh down terms that are extremely common among the corpus, similar to how stop words are removed. The TFIDF method was used as it is the most common of the feature extraction models used in natural language processing and can handle extra words that are not stop words that may be too common across the corpus.

Feature extraction involved the tuning of 2 hyperparameters: n-gram usage and minimum document frequency, the methods for which are detailed in Section IV. Maximum document frequency was also considered at first, but the different values did not seem to have much of an effect on the model, so the best value of 50% of documents was used.

In tuning the n-gram usage, the TFIDF vectorizer takes in a n-gram range *(m,n)* where the feature matrix is built off of n-grams of size *m* up to n-grams of size *n*. For example, the sentence “The quick brown fox jumps over the lazy dog” with an n-gram range of (1,2) would be trained on each word in the sentence as well as each bi-gram such as “The quick”,”quick brown”,”brown fox”, etc.

In tuning the minimum document frequency, the TFIDF vectorizer takes in an integer. The minimum document frequency means the vectorizer will only take into account terms that are in at least **min_df** documents.

D. *Training models*

The models that were tested were: Stochastic Gradient Descent, Logistic Regression, Linear SVM, Polynomial SVM, and Linear Perceptron. The Stochastic Gradient Descent was trained with the logistic and perceptron loss functions, which correspond to Logistic Regression and linear Perceptron training respectively. The poly SVM classifier was tested with polynomials of degrees 2 and 3.

III. Related Work

The first article that helped with this experiment is one that talks about n-gram analysis and various machine learning techniques in analyzing fake news [7]. This article is most closely linked to the experiment done here, and the results are very similar to the results of this project. This paper was an important guide to the experiments done in this project, however it does not do much for background on the subject or data visualization or sentiment analysis. Another paper which was helpful was a review of related work in machine learning methods in fake news analysis [8]. This paper summarized lots of information about the classification of fake news, the different types of fake news, and specifically the efforts of using deep learning methods for developing proper features for training the models. Lastly, the main article useful for the development of this project was by Conroy, Rubin, and Chen [9]. This paper reviewed the main techniques and methods for developing fake news detection at the time of its printing, as well as analysis on fake news as a concept. The article mentions the different approaches in classification, including “bag of words” which was used in this project, deep learning, and sentiment-based classifiers. It also mentions other ways of supplementing classification like networked approaches with linked data, when knowing increased data about topics linked through a network such as social media can help classify text. They had reached the conclusion that analyzing natural language in this way requires multiple layers of analysis and data, in various forms.

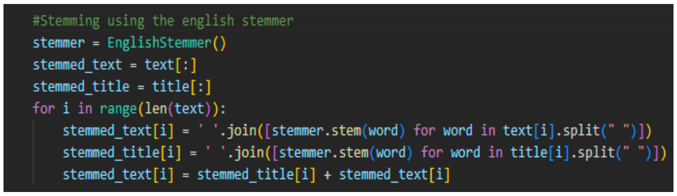
IV. Experiments and Evaluation

A. *Downloading Dataset*

There are 1056 news articles in the dataset, and 816 articles were able to be successfully acquired. The ratio of real to fake articles that were unavailable was about 70% to 30%. Surprisingly, more real articles were unavailable than fake articles. The data was collected into Python lists and assigned a label as 1 for fake news or 0 for real news.

B. *Pre-Processing Data*

The natural language toolkit Snowball stemmer is implemented in its nltk.stem.snowball module [10] (Fig. 1).



```
#Stemming using the english stemmer
stemmer = EnglishStemmer()
stemmed_text = text[:]
stemmed_title = title[:]
for i in range(len(text)):
    stemmed_text[i] = ' '.join([stemmer.stem(word) for word in text[i].split(" ")])
    stemmed_title[i] = ' '.join([stemmer.stem(word) for word in title[i].split(" ")])
    stemmed_text[i] = stemmed_title[i] + stemmed_text[i]
```

Fig. 1 - NLTK english stemmer used in the project. Stems every word in the article text and body and joins it together into one object per article.

The regexp pattern used for the tokenization of the corpus in this project was “**\b[a-zA-Z]{3,}\b”** for which an explanation is in Table I.

TABLE I. REGEXP EXPLANATION

Expression	**Explanation**
\b	Boundary for the word to be matched
[a-zA-Z]	Matches any alphabetic character in lower or upper case
{3,}	Matches the preceding item (alphabetic character) at least 3 times.

Basically, if a sequence of text separated by spaces has at least 3 alphabetic characters in a row, it is considered as a word.

To perform sentiment analysis in Python, the nltk.vader.SentimentIntensityAnalyzer class was used. The analyzer was used on both the pre-stemmed text and the post-stemming text, results can be seen in table II. The compound score from the analyzer was used, which ranges from -1 to 1 in negative to positive.

TABLE II. SENTIMENT ANALYSIS RESULTS

Data		**Sentiment Score**
Pre-Stem	Fake News	0.027
	Real News	0.481
Post-Stem	Fake News	-0.081
	Real News	0.383

Generally, fake news had more negative sentiment and real news had more positive sentiment both before and after stemming. These numbers could fluctuate with a larger dataset and a properly trained sentiment analyzer specific to the dataset, but this aligns with expected trends. Below you can find word clouds of the real and fake news articles to get a top level view of the most common words in each set (Fig. 2).

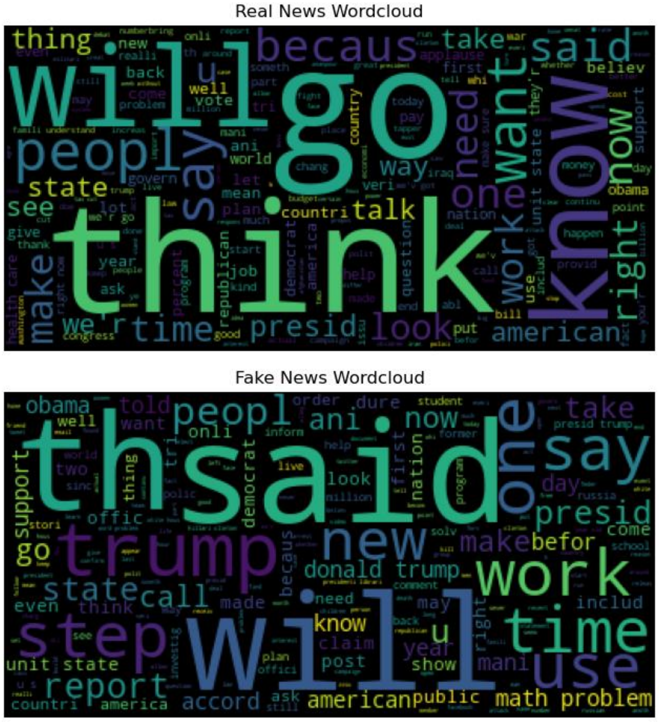


Fig. 2 - Word clouds for fake and real news.

C. Feature Extraction

Stratified k-fold cross validation with 5 folds was used to tune the hyperparameters: n-gram usage and minimum document frequency.

Based on the related work, the n-gram lower bounds tested were 1 and 2, and the upper bounds tested were 1, 2, and 3. This was selected as anything past these numbers the accuracy decreases further and further. The tuning results can be seen below (Fig. 3)

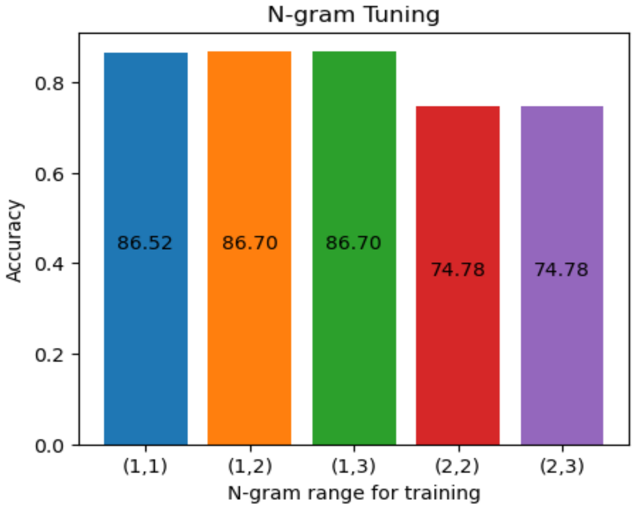


Fig. 3 - The accuracy results in using cross-validation for n-gram range tuning (results plotted with matplotlib).

The best accuracy score changes randomly between the first three ranges depending on the random state seed, so the most important part of the model is that it is at least trained on uni-grams, which if left out decreases the accuracy further.

For minimum document frequency, a range between 10 and 150 documents was tested. These were tested with the maximum document frequency of 50%. The results are below (Fig. 4). The main trend was that increasing the minimum document frequency results in lower accuracy scores, which is expected as the training is being done on less and less data, and important words that do not appear in a lot of documents could get cut out.

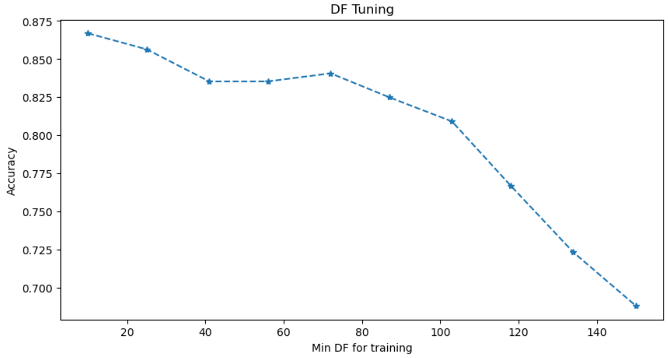


Fig. 4 - The accuracy results in using cross-validation for document frequency range tuning (results plotted with matplotlib).

The vectorizer built a vocabulary based on the training set and fitted to the training set to create a training feature matrix. This was then used to transform the testing set into a test feature matrix. These feature matrices were passed into the classification models detailed previously.

D. Training Models

The models were trained using each respective classifier from sci-kit learn, and predicted labels for the test set were output and plotted (Fig. 5). As expected from the literature review, the linear models performed best in classification, but in this case the polynomial models did not perform that much worse than the linear models.

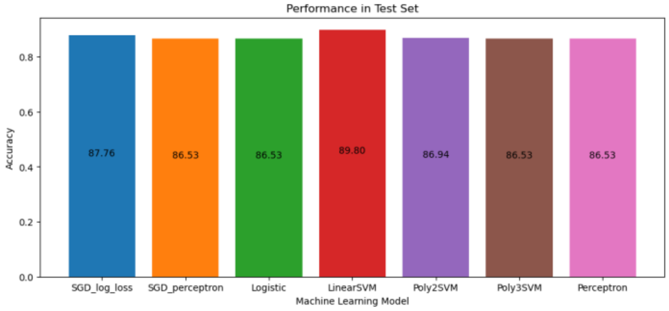


Fig. 5 - The accuracy results in each trained model (results plotted with matplotlib).

For another comparison, the same models above were trained on bi-grams and tri-grams, and it can be seen that the performance is much worse when not including uni-grams (Fig. 6).

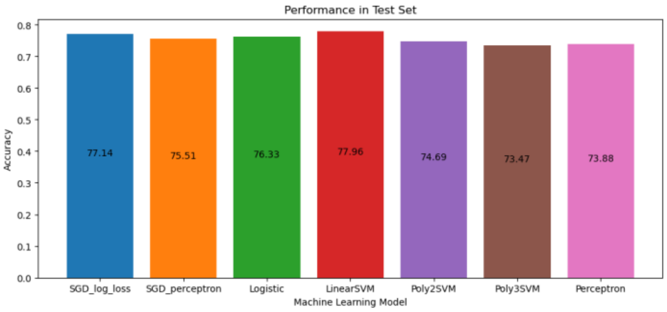


Fig. 6 - The accuracy results in each trained model without including uni-grams (results plotted with matplotlib).

Another trend noticed in the training of the model that in prediction, every model would predict much more false negatives than false positives; an example is below (Fig. 7). This means that the models are much more likely to predict an article isn't fake news when it is, than the other way around. This could be a characteristic of the dataset, as the dataset used for this project was relatively limited, especially with the problems in scraping the articles, with over 200 articles being inaccessible, which is about 20% of the dataset.

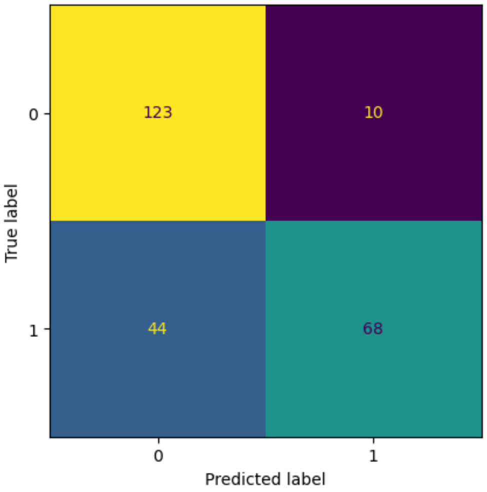


Fig. 7 - The confusion matrix of the linear SVM classifer's predictions.

V. Summary and Conclusions

In detecting fake news, an ample dataset with both fake and real news is beneficial, and with the issues with the current dataset, more data could definitely have alleviated some issues with the project. As expected, training on uni-grams was the most important part of the feature extraction but surprisingly, adding bi-gram or tri-grams did not seem to have much of an effect on the accuracy. Also as expected, the linear models performed better than the nonlinear models, and the minimum document frequency had an inverse effect on accuracy as it increased. Every model predicted far more false negatives than false positives, meaning it passed fake news as real news far more often than the other way around. In future experiments, accuracy and other issues could be solved by simply adding more data, since the dataset was already limited and 20% of the articles were inaccessible. Another way to improve the results in this project without needing much more data could be to add sentiment-based classification to the training and see if there is an added accuracy. With a larger project, it would also make sense to develop deep learning methods for extracting features such as syntax analysis and added networked data.

References

[1] D. M. S. W. D. L. H. L. Kai Shu, "FakeNewsNet: A Data Repository with News Content, Social Context and Spatialtemporal Information for Studying Fake News on Social Media," arXiv.

[2] "Natural Language Toolkit," [Online]. Available: https://www.nltk.org/.

[3] "scikit-learn: Machine Learning in Python," [Online]. Available: https://scikit-learn.org/stable/.

[4] "FakeNewsNet Github," [Online]. Available: https://github.com/KaiDMML/FakeNewsNet. [Accessed April 2023].

[5] "The English (Porter2) stemming algorithm," [Online]. Available: https://snowballstem.org/algorithms/english/stemmer.html. [Accessed April 2023].

[6] [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html#sklearn.feature_extraction.text.TfidfVectorizerature_extraction.text.TfidfVectorizer. [Accessed April 2023].

[7] H. Ahmed, I. Traore and S. Saad, "Detection of Online Fake News Using N-Gram Analysis and Machine Learning Techniques," in *International Conference on Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments*, 2017.

[8] J. S. a. N. S. I. Manzoor, "Fake News Detection Using Machine Learning approaches: A systematic Review," in *3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, Tirunelveli, 2019.

[9] N. K. Conroy, V. L. Rubin and Y. Chen, "Automatic Deception Detection: Methods for Finding Fake News," in *Proceedings of the Association for Information Science and Technology*, 2016.

[10] "nltk.stem.snowball module," [Online]. Available: https://www.nltk.org/api/nltk.stem.snowball.html?highlight=snowball. [Accessed April 2023].