Exploring Fake News Detection Using NLP Techniques

Sri Naga Sai Sushma
Northeastern University

I. Introduction

As widespread dissemination of news through social media platforms has proliferated, it has led to a rise in misinformation. Beginning in 2020, at the start of the Covid-19 Pandemic, rampant misinformation spread which led to negative consequences. To address the widespread misinformation, many companies started to release fact-checkers for the detection of fake news. At this time, many studies were also published to address how to detect whether Covid-19-related news was fake or not.

Our analysis seeks to replicate studies published during this time using the Covid-19 Fake News Dataset. Our goal for the analysis was to assess how different embedding techniques in Natural Language Processing can lead to different outcomes in classification models. We evaluated the differences in word embeddings such as one-hot encoding, bag-of-words, term frequency-inverse document frequency, and Word2Vec. They were evaluated against classical Machine Learning models which include Multinomial Naïve Bayes, Logistic Regression, Support Vector Machine (SVM), Random Forest, K-Nearest Neighbors, Passive Aggressive Classifier, XGBoost, and AdaBoost. Additionally, trained embeddings were explored using a Bidirectional LSTM, and transformers were explored using BERT.

The remaining sections of the paper are arranged in the format that follows. Section 2 introduces the dataset and visualizations used for our analysis. In Section 3, a detailed explanation of our preprocessing methods are outlined. Section 4 contains a description of each model used. In Section 5, results are discussed and models and embedding techniques are compared and evaluated. Section 6 provides a summary and conclusion of the analysis.

II. Dataset

The dataset in consideration deals with COVID-19-related tweet classification as real or fake. We have two files of data which consist of tweets related to COVID-19 and their label as ‘fake’ or ‘real’. The first file which is treated as training data contains around 6,420 tweets out of which 3,360 are labeled as ‘real’ and 3,060 tweets are labeled as ‘fake’. To balance the dataset, the number of fake tweets is up-sampled to match the number of real tweets to 3,360.

The test file consists of 2,140 tweets related to COVID-19 with labels such as ‘fake’ or ‘real’. For evaluating the models, the true labels of the data are compared with predicted labels and assessed using evaluation metrics.

	id	tweet	label
0	1	The CDC currently reports 99031 deaths. In gen...	real
1	2	States reported 1121 deaths a small rise from ...	real
2	3	Politically Correct Woman (Almost) Uses Pandem...	fake
3	4	#IndiaFightsCorona: We have 1524 #COVID testin...	real
4	5	Populous states can generate large case counts...	real

Figure 1. Screenshot of the data

II. Data Visualisation

By visualizing data, we can gain insights into the data. Since we are dealing with tweets, the maximum length of a tweet is 280 characters. Though the tweets are not necessarily 280 characters long, the distribution of the length of the tweets seems bimodal (i.e., the tweets are mostly short and mostly long). The two means of distribution are around 90 and 120 characters, respectively. The mean number of words per sentence is around 17 and they are mostly between 15 and 25 words. The distribution of the number of hashtags per tweet is highly skewed to the right as most of the tweets have no hashtags. By looking at the bar graph of the top 20 words in each class (without the stop words), the frequency of the most common words in both the classes are very similar in comparison. Also, there is no distinct pattern in the top words or in the word cloud.

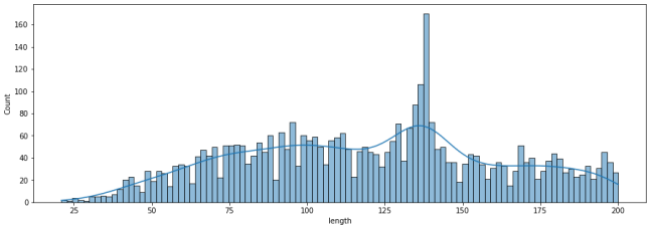

Figure 2. Histogram of Length of characters of a tweet

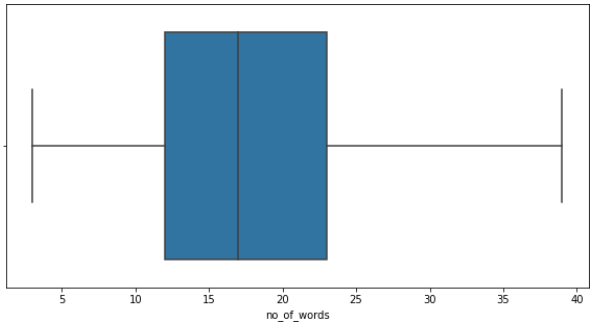

Figure 2. Boxplot of the number of words

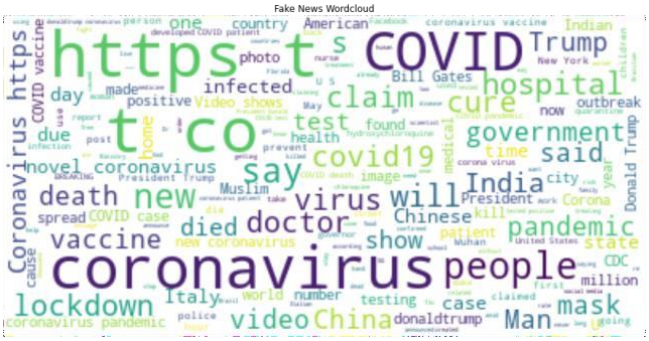

Figure 3. Wordcloud of words in Fake Tweets

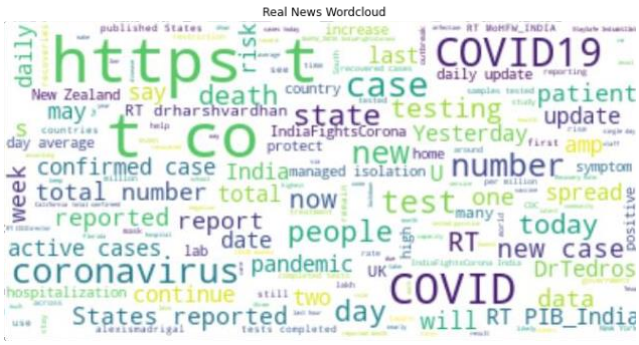

Figure 3. Wordcloud of words in Real Tweets

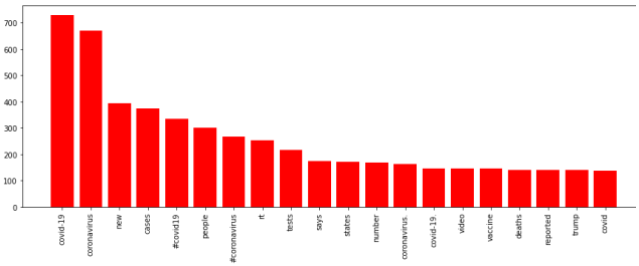

Figure 4. Bar graph of the most frequent words in fake tweets

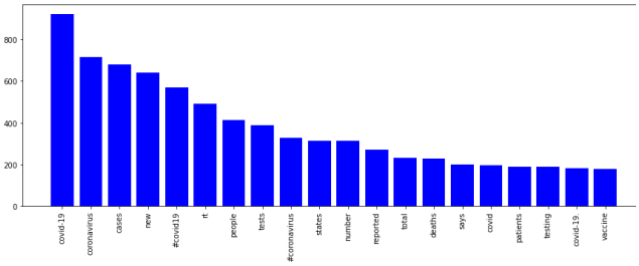

Figure 5. Bar graph of the most frequent words

In Real tweets

III. Data Pre-Processing

We have used the below preprocessing techniques to preprocess the data/text in the column 'tweet'.

1) Stop words are a set of commonly used words in any language. Words like a, an, in, etc. These words are removed from the text as they contribute little but take up a lot of space or our valuable processing time.

2) Convert the text or column into lowercase, as we don't want the "The" and the to be treated separately.

3) Non-ASCII characters like Punctuation, non-English characters, and extra white spaces are removed as they are not relevant to the analysis.

4) Normalization is the process of reducing text into a 'standard' form so that the input is consistent before applying machine learning algorithms. The normalization process contains the following steps:

a) Tokenization: The tokenization splits text strings into smaller pieces such as sentences, or words called tokens. In this project, we will tokenize text into one-word tokens by splitting a sentence by considering white space as a separator. Then, each word is stored in a list which now represents the original sentence

b) Lemmatization: Lemmatization takes into consideration the morphological analysis of the words. To do so, it is necessary to have detailed dictionaries which the algorithm can look through to link the form back to its lemma. We have used WordNetLemmatizer from the nltk package to perform lemmatization.

Word Embeddings

A) **Bag-of-words**: Bag-of-words simply count the frequency of words in a document. Thus, the vector for a document has the frequency of each word in the corpus for that document.

B) **TF-IDF**: In this embedding, the importance of a term is inversely related to its frequency across documents. Term-Frequency is how often a term appears in a document and IDF is about the relative rarity of a term in the collection of documents. By multiplying these values together, we can get the final TF-IDF value. The higher the TF-IDF score the more important or relevant the term is as a term gets less relevant it approaches 0. TF-IDF vectorization calculates the TF-IDF score for every word in your corpus relative to that document and then it puts that information into a vector. Thus, each document in your corpus has its own vector, and the vector has a TF-IDF score for every single word in the entire collection of documents

C) **Word2Vec**: Word2Vec is an algorithm that uses shallow 2-layer, not deep, neural networks to ingest a corpus and produce sets of vectors. Word2vec takes into consideration the context of the words in the corpus. Both continuous Bags-of-words and skip-gram models are implemented. The Skip-gram model takes each word of the large corpus as the input and the hidden or embedding layer using the embedding weights, it predicts the context words.

IV. Methodologies

1. Models

a. Naïve-Baye classifier is a probabilistic machine learning model that's used for classification tasks. The crux of the classifier is based on the Bayes theorem i.e to predict if given a specific word whether a news article is fake or not by obtaining the probability of a fake or real news article given a specific word in the training set. The naive Bayes model assumes independence among features. The GaussianNB model from the sklearn package was applied.

b. Logistic regression is a linear classification model where the decision boundary is made based on a linear function of features i.e it uses input variables (features) to predict a categorical outcome variable (label) that can take on one of a limited set of class values. Hyperparameters such as regularization coefficient C and L1, L2 penalty term, were tuned by the GridSearch method using the sklearn package.

c. Support Vector Machine (SVM) This model is a supervised machine learning algorithm used to classify binary and categorical response data. SVM models classify data by means of optimizing a hyperplane that separates the classes. This can also be considered as finding the hyperplane that maximizes the margin between the classes. SVM models from the sklearn package, linear kernels, are studied in this project.

d. K-Nearest Neighbors Classifier (KNN) K-Nearest Neighbors is a non-parametric supervised lazy-learning algorithm that uses proximity in high-dimensional space to classify labels. It assumes that data points that are close to one another represent the same class. The K-Nearest Neighbors classifier from the sklearn package was applied and trained with the Randomized Search method. The final model selected relied on the 1-Nearest Neighbor for determining class predictions.

e. Random Forest Random Forest is an ensemble learning method that constructs a series of decision trees and relies on the technique bootstrap aggregating. In bootstrap aggregating, bootstrapped samples of data and a random subset of features are used to construct different decision trees. The final classification is determined based on an aggregation of the majority vote from the decision trees. The Random Forest classifier from the sklearn package was applied and trained with the Randomized Search method.

f. AdaBoost: Boosting algorithms work on the principle that the misclassified algorithms by one estimator are given more weight when the training is performed on the successive estimator. For AdaBoost initially, the importance of all data samples is uniformly distributed and based on misclassification or correct classification, each sample's importance is incremented or decremented successively. Coming to the estimators, the error rate of each estimator dictates the weightage of each estimator in the final hypothesis. The higher error rate of a base estimator makes its classification less weighted to give out a final classification for a sample.

g. XGBoost is an optimized distributed gradient boosting (extreme gradient boosting) library designed to be highly efficient, flexible, and portable. It is an ensemble method that has a decision tree as a base estimator. Usually, models tend to alter the weights after every iteration but in the case of using XGBoost, each model is fits on the residual errors of the previous model. For example, for a set of labels 'y', h(x) is the fitted model, and the subsequent model h1(x) would be trained on y-h(x).

h. PassiveAggressive Classifier: This is a type of online learning algorithm which is suitable for large scale data. In this algorithm, due to its online learning nature, the model is updated after encountering each training example. The algorithm exhibits passive nature when the prediction is correct for the current example based on previous training and the model is kept unchanged, whereas the aggressive nature of the algorithm is displayed when there's a misclassification and the model weights are tweaked. This tweaking adjusts the boundaries between classes. This Classifier has a foundation similar to Perceptron but doesn't include a learning rate and has regularization.

i. Bi-Directional Long Short Term Memory (LSTM) A bi-directional LSTM is a type of RNN that is more robust to the vanishing gradient problem and can learn long-term dependencies. An LSTM model has feedback connections that allow it to process sequential data that have a temporal aspect. The bi-directional layer in the LSTM allows information in the model from the past and future to flow to the present. This is useful as past and future information in text can influence outcomes. A bi-directional LSTM was built using the Keras package. The model consisted of an embedding layer, a bi-directional layer, and two dense layers followed by a dropout layer. The model was tuned using Keras RandomSearch Tuner. The hyperparameters tuned included learning rates, batch sizes, units, dropout percentages, l1 regularization, l2 regularization, and optimizers such as Adam, Stochastic Gradient Descent, and Root Mean Squared Propagation. However, due to the limited size of our data and the

number of observations required for deep learning to be successful, the model suffered from severe overfitting and was not included in the final analysis.

j. Transformers: An effective way to achieve high performance on the classification of sequence data can be achieved by fine-tuning a large pre-trained model. BERT (Bidirectional Encoder Representations from Transformers) is a deep learning model that is trained on large datasets. It is a combination of two neural networks: a transformer encoder and a transformer decoder. The BERT model is trained on large datasets such as Wikipedia and uses a combination of word embeddings and a combination of deep neural networks to represent text. It is very effective at extracting meaning from text. A bert-uncased-base from hugging face's transformer library was used with PyTorch as the backend. Additional layers were added to the model for fine-tuning this model whilst freezing the pre-trained weights of bert.

Model	**Optimal Parameters and hyper parameters chosen**
Naive Bayes	Variance Smoothing =0.001233
Logistic Regression	inverse of regularization coefficient C=0.0001, Penalty= L2, solver = liblinear
SVM	C=1, Kernel= rbf, gamma=0.01
BERT	Batch size = 32, Max Sequence Length = 50, optimizer=adam(lr = 2e-5), FF + dropout(0.5) +FF, epoch = 3

Table 1. Optimal Parameters

	[Accuracy, Precision, Recall, F1]								
	RF	**KNN**	**LR**	**NB**	**SVM**	**BERT ***	**ADA Boost**	**XGBoost**	**PAC**
1-0	[0.89, 0.86, 0.91, 0.88]	[0.71, 0.74, 0.61, 0.67]	[0.77 , 0.64, 0.84, 0.73]	[0.90, 0.92, 0.88, 0.9]	[0.91, 0.91, 0.91, 0.91]	[0.94, 0.865, 0.85, 0.855]	[0.81,0.8,0.71,0.77]	[0.82,0.76,0.9,0.83]	[0.9,0.91,0.87,0.89]
BoW	[0.89, 0.86, 0.91, 0.89]	[0.71, 0.73, 0.63, 0.67]	[0.76 , 0.62, 0.84, 0.72]	[91.9, 0.87, 0.94, 0.92]	[0.92, 0.91, 0.92, 0.91]	[0.94, 0.865, 0.85, 0.855]	[0.81,0.8,0.71,0.77]	[0.82,0.76,0.9,0.83	[0.9,0.91,0.97,0.89]
TF-IDF	[0.7,0.8 , 0.82, 0.70, 0.76]	[0.89, 0.91, 0.85, 0.88]	[0.85 , 0.78, 0.89, 0.83]	[88.13, 0.89, 0.85, 0.87]	[0.92, 0.94, 0.89, 0.65]	[0.94, 0.865, 0.85, 0.855]	[0.8,0.81,0.72,0.77]	[0.82, 0.72, 0.9, 0.83	[0.91,0.72,0.97,0.9]
Cont (Word2Vec)	[0.79, 0.84, 0.69, 0.76]	[0.75,0.77 , 0.69, 0.73]	[0.53 , 0.46, 0.62, 0.46]	[0.48, 0.52, 0.46, 0.48]	[0.48, 0.99, 0.48, 0.65]	[0.94, 0.865, 0.85, 0.855]	[0.69,0.71,0.6,0.65]	-	[0.68,0.62,0.88,0.73]
SG(Word2Vec)	[0.76, 0.84, 0.61, 0.71]	[0.73, 0.81, 0.57, 0.67]	[0.52 , 0.44, 0.47, 0.46]	[0.48, 0.43, 0.46, 0.49]	[0.48, 0.47, 0.46, 0.47]	[0.94, 0.865, 0.85, 0.855]	[0.69,0.83,0.59,0.64]	-	[0.74,0.74,0.72,0.73]

Table 2. Metrics vs Pre-processing Techniques

RESULTS AND CONCLUSION

In conclusion, the outcomes of our results suggest that traditional embedding techniques such as one-hot encoding, TFIDF, and a bag of words generally outperform more advanced techniques such as Word2Vec. Similarly, we observed that traditional machine learning models were able to achieve similar performance to more advanced transformer models like BERT for the classification task. Our observations showed that the embedding technique with the most success was highly dependent on model choice. While most models in our analysis had comparable performance in classifying Covid-19 fake news, the model that performed the best was the Support Vector Machine with the Bag of Words embedding technique achieving an accuracy, recall, and f1 score of 92%, 92%, and 91%, respectively.

In BERT, the performance does not improve the results with pre-processing and lemmatization. The AUC remained consistent but the F1 score decreased to 0.835 from the original value of 0.855. As BERT and its family of models learn to compute text representations in context. In the sense, that the representations computed for a word in a specific sentence would be different from the representations for the same word in a different sentence. This context also comprises stop words and punctuation which can very much change the meaning of a sentence. When the max sequence length is set to 50, the model performs the best. However counterintuitively, with the increase in max sequence length, the model performs worse. With an increment of 25 to the max sequence length, the model optimized the recall of a single class.

Our analysis was more concerned with achieving a higher recall as with fake news detection, we are more concerned with reducing the false negatives than the false positives. Given that fake news can lead to harmful consequences, it is essential that we capture as many fake news articles as possible at the expense of accidentally classifying a real article as fake or observing a false positive. However, a trade-off exists between precision and recall, therefore our goal was to identify the model with the highest recall and F1 score.

We theorize that the Support Vector Machine resulted in the best performance as SVM models were considered state of the art prior to the evolution of neural networks. As our dataset was small and not conducive to deep learning, the SVM appears to have been able to deduce the complex non-linear relationships that exist in language. The SVM also probably observed better performance than a more complex model such as BERT as it was trained on the specific topic of Covid-19. Additionally, we theorize those traditional embedding techniques performed better as Word2Vec relies on a neural network architecture which was not conducive to our small datasets as we had fewer observations than necessary for a neural network to be successful.

Further improvements to the project could be made by gathering more data. As Covid-19 fake news was very prevalent in the past 2-3 years, numerous datasets exist on the topic which would allow us to train better models. Additionally, augmentation techniques such as back-translation could be used in future iterations to build a larger dataset from which more insights could be discovered.