Covid-19 Misinformation Classification Systems:

An Analysis of Binary Classifiers to Detect U.S. Covid-19 Misinformation in Social Media and News Articles

Sarah Gonzalez

Stephanie Myott-Beebe

Anel Nurkayeva

Shiran Zhang

SIADS 697/698: Capstone

University of Michigan School of Information

Abstract

Despite the sheerly overwhelming availability of news, staying informed is more difficult than ever in today's climate because the reader must constantly question whether a piece of news is accurate or not; whether it is true or has been written to intentionally mislead the reader. This is particularly salient with news regarding the global COVID-19 pandemic. Notwithstanding efforts by many to detect and prevent COVID-19 misinformation, it persists. The authors of this paper review past research, compile data from several sources, clean and preprocess the data, and analyze a number of machine learning and deep learning binary classification methods–logistic regression, random forest, gradient boosting machine, ensemble method, long short-term memory, BERT, and RoBERTa–to find the optimal approach to combat the negative effects of misinformation campaigns. The authors conclude that the best performing approach by far is a RoBERTa model trained with autoML. Going forward, with more time and resources, the authors are interested in creating a better, more robust dataset and increasing hyperparameter tuning efforts to maximize performance and the ability of a model to accurately detect COVID-19 misinformation.

Covid-19 Misinformation Classification Systems:
An Analysis of Binary Classifiers to Detect U.S. Covid-19 Misinformation in Social Media and News Articles

As the coronavirus disease 2019 (COVID-19) pandemic has spread globally, so too have rumors, fears, discrimination, and stigma become increasingly pervasive on social media. False information about the epidemic has caused psychological anxiety and social panic, deepened racial and geographic discrimination, reinforced stereotypes, exacerbated antagonism and conflict, and hindered communication and cooperation on a global scale. The proliferation of misinformation about COVID-19 has made it difficult to find trusted sources of information and guidance to rely on, potentially impeding disease control and containment with life-threatening consequences. Identifying misinformation about COVID-19 is therefore a necessary, albeit daunting, task.

According to Siwakoti et al. (2021) in "Localized Misinformation in a Global Pandemic: Report on COVID-19 Narratives around the World," fake news about COVID-19 is highly heterogeneous across regions and countries, with prominent themes varying widely from region to region, and from country to country. For example, misinformation in sub-Saharan Africa focuses on government responses to the pandemic and singing praises for African leaders. Meanwhile, misinformation in Europe typically involves false rumors about both governmental and nongovernmental organizations. A prominent theme in China is spreading misinformation about individuals' COVID-19 diagnoses. In the United States, COVID-19 misinformation often arises out of, and tends to exacerbate, political polarization surrounding the disease.

Given that themes of COVID-19 misinformation vary so significantly from one country to the next, a thorough global analysis of the subject would require an exhaustive amount of time and technology. Unfortunately, our limited resources do not afford us the option to undertake such an extensive analysis. Considering that the United States is one of the countries most

affected by the pandemic, for our purposes, we have chosen to analyze COVID-19 news released primarily by the U.S. media. Specifically, the purpose of our project is to create and analyze binary classifiers to identify a piece of U.S. COVID-19 news as either real or fake.

The remainder of this paper is structured as follows: First, we conduct a literature review of past research on binary fake news classifiers. Next, we describe our approach to collecting, validating, and cleaning the text of our dataset, which consists of a combination of social media content and news articles published by major U.S. media about the COVID-19 pandemic. The following section discusses the machine learning and deep learning models for binary classification that we analyzed with respect to our dataset, including Logistic Regression, Random Forest, Gradient Boosting Machine, Ensemble, Long Short-Term Memory (LSTM), Bidirectional Encoder Representations from Transformers (BERT), and Robustly Optimized BERT Pretraining Approach (RoBERTa). Thereafter, we discuss the results of our analyses and what they mean for the detection of COVID-19 misinformation. We conclude the paper by highlighting some of the limitations of our study as well as the areas upon which we would like to improve and/or perform additional work in the future.

Literature Review

Previous research on misinformation classifiers has focused on a number of different approaches in both the models used and the type of data analyzed. This literature review is not exhaustive but provides excellent examples of some of the exciting work being done in this area.

Fake News Detection Using Text

Some approaches to misinformation classification analyze text alone. For example, in their paper “Twitter Rumour Detection in the Health Domain,” Sicilia et al. (2018) test a number of approaches to fake news detection specifically in the health domain. Although the paper is relatively new, it is one of the earlier works to address this topic. The authors test a number of different features, including influence potential, personal interest, and network characteristics,

as well as a number of different models, including multi-layer perceptron, nearest neighbor, support vector machine, random tree, multiclass Adaboost, and random forest. They use default parameters for the various models to test generally how one model might perform compared to another, ultimately concluding that the random forest approach shows the best results.

Nasir et al. (2021) rely on deep learning methods in "Fake News Detection: A Hybrid CNN-RNN Based Deep Learning Approach," using a combination of a Convolutional Neural Network (CNN) and a Recurrent Neural Network (RNN). Their focus is on providing a solution to "the lack of thorough investigations on relevantly new datasets and the lack of combinations of deep learning models or neural networks for fake news detection" (Nasir et al., 2021, p. 4). Using publicly-available English documents with a mix of real and fake news, the authors create a deep learning model using the Keras library. Their model contains the following layers: (a) a word embedding layer, (b) a CNN layer to extract local text features, (c) a pooling layer, (d) a RNN (in the form of a LSTM) layer, and (e) a dense layer to shrink the output space to 1-dimension for a fake or real classification label. Nasir et al. (2021) suggest that their approach could push research forward with respect to the generalizability of fake news detection.

In their paper "A Novel Rumor Detection Algorithm Based on Entity Recognition, Sentence Reconfiguration, and Ordinary Differential Equation Network," Ma et al. (2021) test their misinformation classification algorithm on Tweets. The text to be analyzed includes the Tweets themselves as well as comments and retweets. The authors take a unique approach by reconfiguring the text before feeding it into the model; they defend their approach by stating that the "difficulty of this research lies in the disambiguation of entity meanings, which is important in the short text without enough context" (Ma et al., 2021, p. 224). Before performing any analysis, they process the text with entity recognition and sentence reconfiguration to account for meaning that can be lost with the short text of Tweets. The authors find promising

results from this approach and suggest that entity recognition and sentence reconfiguration should become a more standard feature of fake news detection.

Fake News Detection Using Text and Images

Unlike the previous Section, some misinformation classification approaches analyze images in addition to text. The authors of the paper "Detection and Veracity Analysis of Fake News via Scrapping and Authenticating the Web Search" discuss their proposed algorithm to "analyze the veracity of the news events that are floating in the form of images in social media" (Vishwakarma et al., 2019, p. 221). Vishwakarma et al. (2019) contend that their approach is unique because it works on images, but they are essentially analyzing text in image format; that is, they are pulling text contained in images using optical character recognition (OCR), then evaluating that text for its veracity. However, this approach is unique in that rather than relying upon a machine learning or deep learning model, the model authenticates text with web searches. Specifically, the algorithm extracts entities from the text, searches for those entities on the web, and scrapes content from the links that it deems reliable. The algorithm then uses the scraped content to determine whether the news is real or fake. The paper shows promising results for the future of fake news detection using web scraping.

Finally, newer research is advocating an approach that analyzes text in conjunction with its co-occurring images. In their paper "Fake News Detection for Epidemic Emergencies via Deep Correlations between Text and Images", Zeng et al. (2021) advocate for an approach to fake news detection that analyzes both text and images. They indicate that given the prevalent co-occurrence of images with text, as well as the semantic correlations between co-occurring text and images, text analysis on its own is no longer sufficient in the fight against fake news. The authors use a novel end-to-end deep neural-network-based method that learns multimodal fusion representation and an image-augmented text representation. They find that this hybrid approach has better performance than other well-performing fake news detectors. With future

work on fake news detectors, they suggest that the more emphasis is placed on the correlation between text and images, the better, and that more work should be done on analyzing images themselves because it is becoming increasingly easier to create fake images.

The Data

Obtaining and Compiling the Data

The data for this analysis is a mixture of data found and classified in other professional studies and our own data scraping. Although we initially were interested in analyzing Tweets as well, our final approach was to focus primarily on news articles.

The first professional dataset that we pulled data from was "CoVID19-FNIR". The authors of the dataset describe it as follows:

> [The dataset] contains news stories related to CoVID-19 pandemic fact-checked by expert fact-checkers. CoVID19-FNIR is a CoVID-19-specific dataset consisting of fact-checked fake news scraped from Poynter and true news from the verified Twitter handles of news publishers. The data samples were collected from India, The United States of America, and European regions and consist of online posts from social media platforms between February 2020 to June 2020.

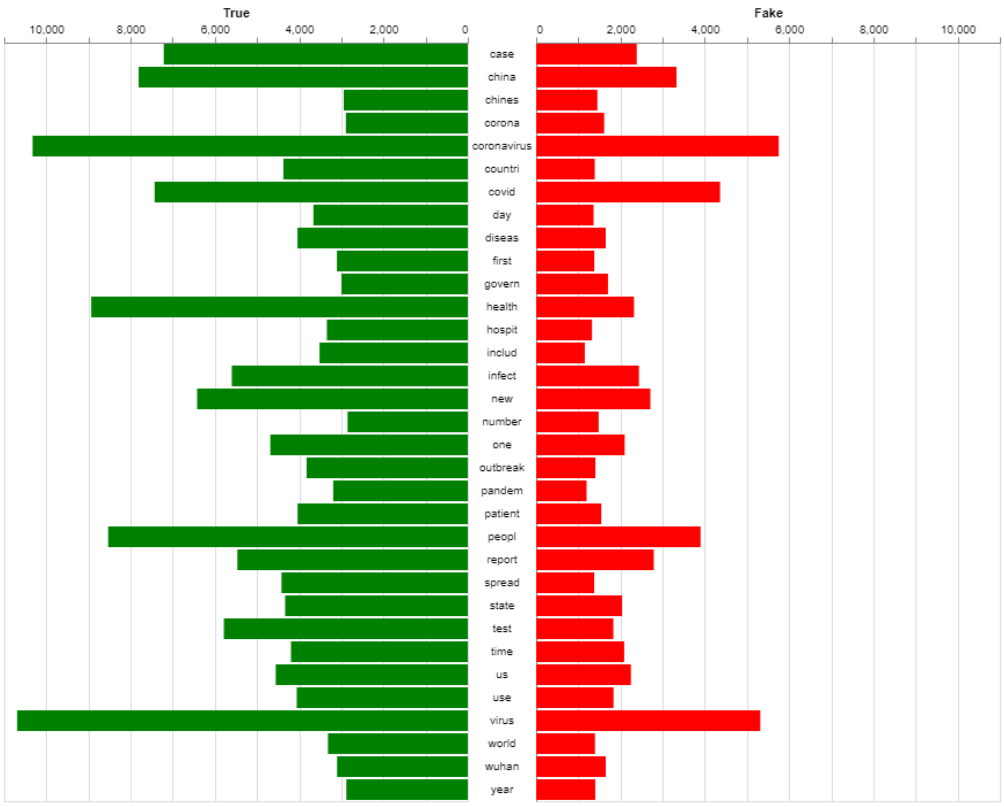
(Saenz et al., 2021). Given that the articles labeled "TRUE" were Tweets, we dropped them because we were interested in classifying news articles and not Tweets. This gave us about 3,700 false entries. Next, we added articles from the "COVID-19 Fake News Dataset", which was manually labelled to "true," "false," and "partially false"; we considered articles labeled "partially false" to be false. (Koirala, 2021). This gave us 2,601 true news articles and 4,439 false articles. We also included 1,504 English language articles from the "ESOC COVID-19 Misinformation Dataset" that were themselves true, but about fake news. Examples include articles titled, "No, the CDC hasn't stopped calling COVID-19 a pandemic" and "Wisconsin Assembly speaker says state passed one of the first COVID-19 bills, but that's wrong"(Shapiro et al., 2020). These

articles were crucial to add because one of the focuses of our project was creating a model that correctly classified true articles about fake news.

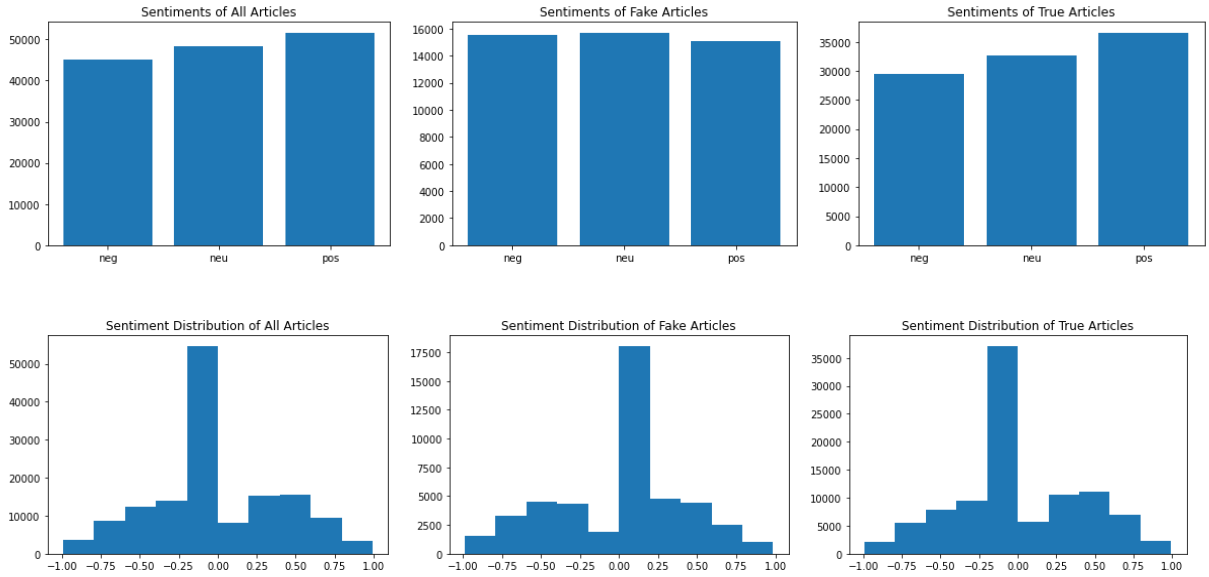
Finally, we wanted to include articles in our dataset from around the time that we ran our study (December 2021). We scraped the web using NEWSAPI and the keyword “covid” to access a variety of English language articles. We also scraped articles from zrohedge.com, infowars.com, and gateway pundit, which, after manual labeling, turned out to be all misleading.

Validating the Data

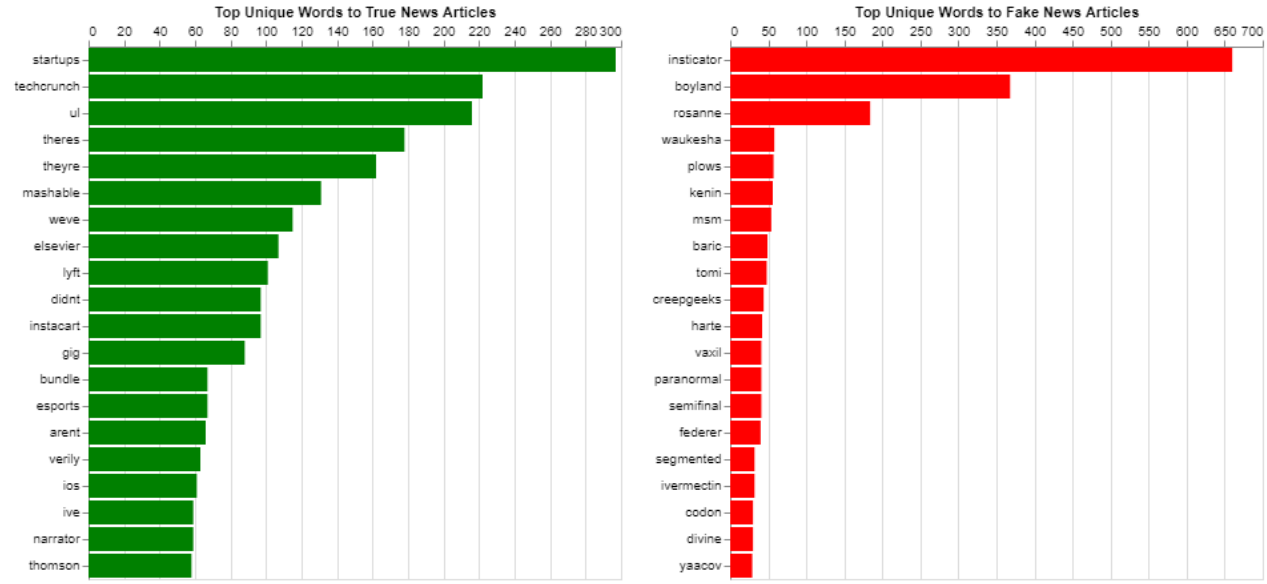
Our final dataset contained 4,987 true news articles and 5,600 fake news articles. An important thing to note is the variation in article length: The average length of a true news article is 417 words, and the average length of a fake news article is 211 words. This discrepancy could result in some overfitting of the models. However, while validating the dataset, we did not see a substantial difference in the ratio of words between the two groups. The table below shows the most common words that occur in both true and fake news articles:



We ran a quick sentiment analysis on our dataset (see charts below). An interesting aspect of the articles is that they have the same general distribution of sentiment. That said, most fake articles had a neutral, slightly more negative sentiment than true articles, which generally were more positive than negative. While we did not run any analysis on whether these findings were statistically significant, it is insightful nonetheless and warrants more study.



Another method we used to validate the integrity of our data was to test the words (and their frequencies) used exclusively in true versus fake news articles. In our first iteration of the dataset, we realized that most of the higher frequency words exclusive to true news included the names of Canadian cities and provinces. As such, in our second iteration, we adjusted the web scraper to look for U.S.-based news *sources*, as opposed to simply U.S.-based news articles. We noticed that one of the most frequent words unique to the true dataset was “booster”, suggesting that all our fake news articles were written pre-April 2021, before healthcare professionals were even talking about the Covid-19 booster shot. To remedy this, in our third iteration, we scraped data from unreliable news sources, such as infowars.com, and manually labeled the false articles. The following tables show the top unique words for true and fake articles in the dataset that we used to train our models (please note the differences in axis values):



We wanted to create a COVID-19 misinformation binary classifier that was robust to different types of news articles, which meant combining articles from a number of different sources. As discussed in our "Moving Forward" Section, below, with more time and resources, we would expand upon and further curate this dataset. However, given the various sources from which we pulled our data, we feel that overall, our dataset is unique and has a decent amount of more current news entries compared to other sources.

Cleaning and Preprocessing the Data

We were somewhat aggressive with cleaning and preprocessing the data before feeding it into our models. In particular for models that took longer to train, e.g., LSTM, we wanted text that was as clean as possible so as to reduce dimensionality and increase effectiveness as well as computational efficiency. Specifically, we cleaned our data using the following steps:

- Basic text cleaning: We performed basic text cleaning by making all the text lowercase and removing punctuation, numerical digits, and unicode characters.
- Stop word removal: We removed stop words from the text using NLTK's list of English stop words. A stop word is one that occurs so frequently that it adds little

value for purposes of textual analysis. Splitting our text for stop word removal also removed any unnecessary whitespace characters.

- Rare word removal: We removed rare words (which we defined as any word that occurred only once in the entire corpus) from the data. Words that only appeared once could not add much meaning to the models, so removing them allowed for some dimensionality reduction in the long-term.
- Stemming: We stemmed the data with nltk's SnowballStemmer. Stemming reduces a word to its base stem, enabling words with the same stem (and thus very similar meanings) to be represented with the same word in the cleaned data. Stemming was also helpful to reduce the number of unique words in the dataset.
- Removal of short documents: We concluded our data cleaning by removing any rows where the text was less than three words long, concluding that these documents would add little meaning to the models.

After cleaning the text, we used Keras's Tokenizer class to transform it into a format that our models (in particular, our LSTM model) could understand and utilize. Each unique word was assigned an integer, then each sample was transformed into a sequence of applicable integers. The Tokenizer class also allowed us to extract the total vocabulary of words used in our dataset. Once the text was transformed into integer sequences, we split the data into training (70%), validation (15%), and test (15%) datasets using Scikit-learn. We intentionally split the data after we had tokenized it because we wanted to ensure that we were using a single vocabulary for all our datasets. If we separately tokenized our training, validation, and test datasets, we could run into an issue of words appearing in one set but not in another, as well as of different integers being assigned to the same word in different datasets.

Methods and Models of Binary Classification Tested

This Section discusses the various binary classification models that we used in our analysis of U.S. COVID-19 misinformation.

Logistic Regression

A logistic regression model predicts the probability of a binary outcome–labels a sample as real or fake news–given a set of independent variables. Logistic regression was the baseline against which we compared our more complex, deep learning models.

Logistic Regression Models Analyzed

We trained several logistic regression classifiers using the following increasingly complex approaches:

- Most frequent dummy: A most frequent dummy classifier labels every item with the label that appears most frequently in the data, which for our data is “fake”.
- Uniform distribution dummy: A uniform distribution dummy classifier randomly generates predictions, giving equal weight to both “true” and “fake” labels.
- Bag of words (BoW): A BoW vectorizer represents textual data by word counts, or by the number of times a word appears in a document, without giving any importance to word order.
- N-gram(1,2) vectorizer: An n-gram vectorizer extends the BoW vectorizer with the counts of the applicable n-grams; here, bi-grams (or two adjacent words).
- N-gram(1,3) vectorizer: This approach builds upon the previous one by including the counts of tri-grams (or three adjacent words).
- TF-IDF vectorizer: Rather than the pure word count of a BoW, a TF-IDF (“term frequency-inverse document frequency”) vectorizer uses word count normalized by the number of documents in which the word appears.

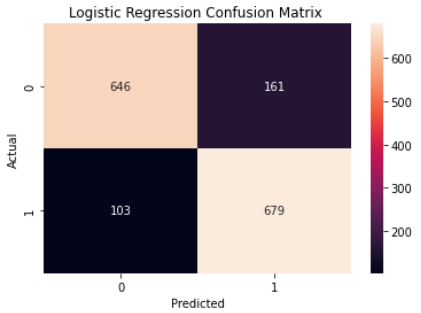
- Gensim vectorizer: Gensim is a python library used for representing documents as semantic vectors (not unlike a TF-IDF vectorizer).

Hypertuning Parameters for the Logistic Regression Models

In order to ensure that the logistic regression models performed optimally, we used Scikit-learn's GridSearchCV to hypertune parameters. Based on the results, we determined that the following parameters maximized performance:

Parameter	**Description**	**Best Value**
C	Strength of regularization to apply to the model to prevent overfitting	0.1
solver	Algorithm to use with optimization; related to the choice of penalty parameter	"saga"
max_iter	Maximum number of iterations for convergence	10000
penalty	Penalty used to regularize the model and prevent overfitting	"l2"

Results from Logistic Regression Models



We received a wide range of results from the various logistic regression vectorizers, from an F1-score as low as 34.58% (with the most frequent dummy and Gensim vectorizers) to one as high as 83.37%. The best F1-score came from the n-gram classifier trained on single words and bi-grams only (ngram(1,2)). The scores for this model are detailed in the table below:

	Precision	**Recall**	**F1-Score**	**Support**	**Accuracy**
Label = 0	0.86	0.80	0.83	807	
Label = 1	0.81	0.87	0.84	782	
Total					0.83

Surprisingly, we had very little success with the Gensim vectorizer. With additional time and resources, we could have performed more analysis to improve performance of the Gensim model, but for the current project, we felt that our time was best spent training other models.

The following table contains the results of all our logistic regression models:

Vectorizer	**F1 score**
Most Frequent Dummy	34.58%
Uniform Distribution Dummy	49.49%
Bag Of Words	82.56%
Ngram(1,2)	83.37%
Ngram(1,3)	82.93%
Tf-idf	78.57%
Gensim	34.58%

Random Forest

The second baseline model that we used was a random forest. In order to understand a random forest model, one must first understand a decision tree classifier. A decision tree is a classifier (though it can also be used for regression) resembling a flowchart of many branches of hierarchical nodes that distributes data according to “tests” on attributes in the data. Each of the terminal nodes (called “leaf nodes” in a decision tree) represent a class label. One classifies a new data point by applying the decision tree tests and following the applicable branches until one reaches a leaf node. A random forest is a collection of said trees created randomly and under certain constraints that work in unison to classify a data point. The goal is that the errors of an individual tree will be minimized by the forest as a whole, thus creating a much more accurate and robust classifier than a simple, single decision tree ever could.

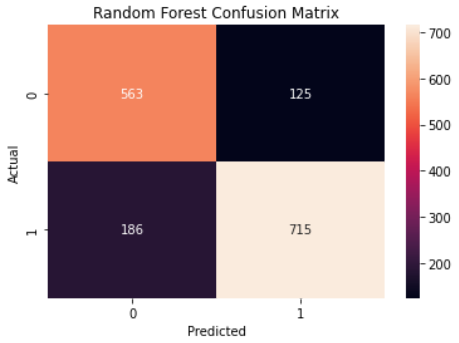
Hypertuning Parameters for the Random Forest Model

Unlike with the logistic regression model, we trained our random forest model only using the ngram(1,2) vectorizer. We hypertuned certain parameters of the model using GridSearchCV. Our best results were obtained with the following parameters:

Parameter	Description	Best Value
max_depth	The maximum depth of a single decision tree	80
n_estimators	The number of trees to include in the random forest	100

Results from the Random Forest Model

The best accuracy that we reached for our random forest model was 80.25%, with an F1-score of 78% for samples labeled 0, and 82% for samples labeled 1. The table below summarizes our results. Although the results were disappointing, we continued to test other models to see if we could enhance the model's performance.



	Precision	Recall	F1-Score	Support	Accuracy
Label = 0	0.75	0.82	0.78	688	
Label = 1	0.85	0.79	0.82	901	
Total					0.80

Gradient Boosting Machine

The third model that we tested was a gradient boosting machine, which is an ensemble method that creates a strong predictive model from many weak learners. There are many types of gradient boosting machines, but for our analysis, we used a simple GradientBoostingClassifier (GBC) from sklearn.ensemble. Similar to a random forest, a GBC uses many decision trees to

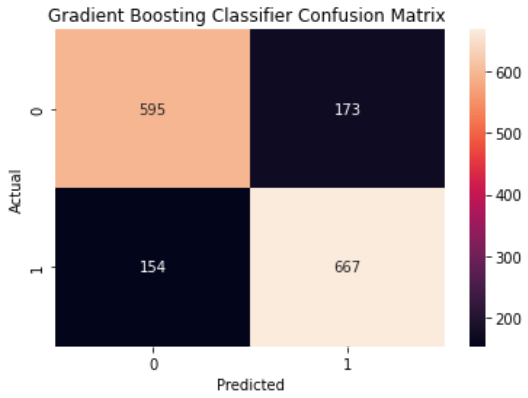
classify a sample. However, unlike a random forest, a GBC is built iteratively by improving upon the errors of the previous model using a specified loss function for its predictive power.

Hypertuning Parameters for the Gradient Boosting Machine

Like our random forest model, only our ngram(1,2) vectorizer was used to hypertune parameters. In order to find the best parameters for our model, we again used GridSearchCV. Our best results were obtained with the following parameters:

Parameter	**Description**	**Best Value**
n_estimators	The number of boosting stages to improve upon the model's performance	100
learning_rate	The amount by which to shrink the contribution that each tree makes to the model	1.0

Results from the Gradient Boosting Machine

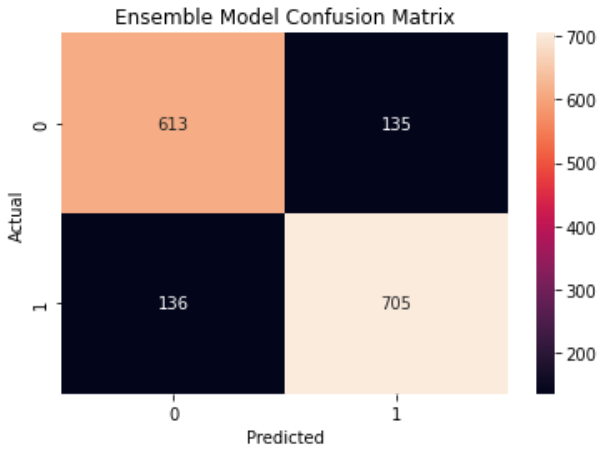


The best accuracy score achieved by the gradient boosting machine was 79%, with an F1-score of 78% for samples labeled 0, and 80% for samples labeled 1. Again, these results were not as promising as we had hoped to see, but in our next model we tried to minimize the errors found in the previous ones. The results are in the following table:

	Precision	**Recall**	**F1-Score**	**Support**	**Accuracy**
Label = 0	0.79	0.77	0.78	768	
Label = 1	0.79	0.81	0.80	821	
Total					0.79

Ensemble Model

The final "simple" model that we tested was an ensemble model using VotingClassifier in Scikit-learn. An ensemble model is not a model in itself, but rather a mixture of models that takes the most common classification from each model and chooses the most likely label. In classifying a sample, an ensemble uses one of two types of voting: The first type of voting is "hard" voting, which makes a prediction based on the majority of predicted class labels; thus, if two models predict an article is fake, and one predicts true, the final label will be "fake". The second type of voting, "soft" voting, uses the argmax of the sums of the predicted probabilities in order to make the best prediction.



Our ensemble included all three models discussed above–logistic regression, random forest, and gradient boosting machine–with the same hypertuning of parameters contained in those models. The ensemble approach enabled us to achieve a score of 83.44% with hard voting and 83.88% with soft voting (our highest accuracy score yet).

	Precision	**Recall**	**F1-Score**	**Support**	**Accuracy**
Label = 0	0.82	0.82	0.82	748	
Label = 1	0.84	0.84	0.84	841	
Total					0.83

Long Short-Term Memory (LSTM)

A LSTM model is an improvement upon a Recurrent Neural Network (RNN), which is a form of Neural Network (NN) for sequential data. A RNN is commonly used to model language,

which can be considered a sequence of words. A feedforward vanilla NN treats inputs and outputs as independent of one another, and as such starts fresh with each new iteration of training the model; it does not remember what it learned in a prior iteration. Unfortunately, this often does not lead to good model performance with text data, in which the meaning of a word within a sentence very much depends upon the words that preceded (and that follow) it.

Unlike a feedforward vanilla NN, a key component of an RNN is sequential memory; specifically, the current layer in a network will be informed in part by information contained in prior layers. This is crucial when dealing with textual data, as the meaning of a piece of text is often informed not by words in isolation, but rather by their interaction with and relationship to other words. A word may take on a different meaning depending on the words that surround it. Despite the fact that the RNN is designed to handle sequential memory, an issue that often occurs is vanishing gradient, in particular with longer sequences; that is, an RNN can have difficulty remembering what happened early on in a long sequence.

LSTM is designed to address the short-term memory limitations of a typical RNN. A LSTM layer includes an input layer, a hidden layer, and an output layer, with the hidden layer containing memory cells and gate units. Specifically, a LSTM cell has the following gates: forget gate, input gate, and output gate. The forget gate decides whether information from a previous step is relevant, and thus whether it should be remembered or forgotten. The input gate allows the LSTM to learn new information from the input. The output gate pushes the new information (including any updates) to the next step. In this process, relevant information is retained in memory regardless of the number of timesteps.

Constructing LSTM Model

We utilized the TensorFlow Keras library to construct our LSTM model using the following layers:

- Embedding layer: The embedding layer transforms text (represented by sequences of integers) into word vectors in a continuous vector space. The relevant parameters of the embedding layer include the following:
  - *input_dim*: Size of the vocabulary to be embedded.
  - *output_dim*: Dimension of the output vector.
  - *input_length*: Constant length of each input sequence.
- Dropout layer: The dropout layer prevents the model from overfitting. A dropout layer works by randomly setting each input value to 0 at the specified frequency at each step in the training process. When selecting an optimal rate for the dropout layer, a balance must be struck between retaining accuracy of the model while at the same time preventing the model from overfitting. We used the following parameter in this layer:
  - *rate*: Fraction of the input units to set to 0.
- LSTM layer: This is the layer that actually performs the LSTM algorithm. As discussed above, a LSTM model is a form of RNN that analyzes textual data while correcting for the vanishing gradient problem that can arise with a standard RNN. We used the following parameter in this layer:
  - *units*: Dimension of the output.
- Dropout layer: We added another dropout layer after the LSTM layer to further prevent the model from overfitting.
- Dense layer: The last layer before a prediction is made is a fully-connected dense layer, in which every neuron is connected to every neuron in the preceding layer. The dense layer transforms the input neurons into the output dimension using an activation function.

- *units*: Dimension of the output, which should be 1, i.e., prediction as to whether a piece of COVID-19 news is fake or not.
- *activation*: Activation function to transform the inputs into the output. We used a 'sigmoid' activation function, which is standard in LSTM models.

Once we added all the necessary layers to our model, we compiled it prior to training using the following parameters:

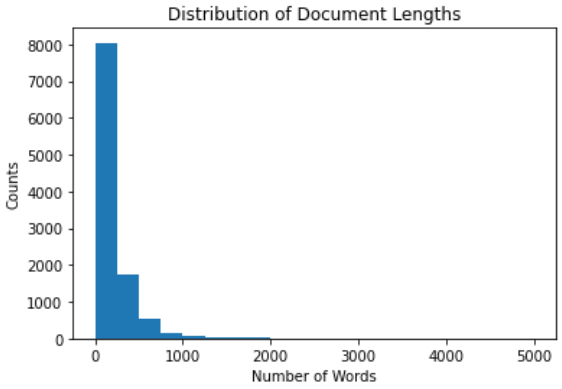
- *optimizer*: When a model is trained, its weights are initially set randomly. During each epoch, the weights are updated so as to increase accuracy and decrease loss. The optimizer is a method to optimize the model's loss function. We used the *Adam* optimization algorithm, which extends the stochastic gradient descent algorithm by using adaptive learning rates and momentum to converge more quickly.
- *loss*: The loss function in a deep learning model is the loss that the model is seeking to minimize. We used a binary cross-entropy, which is the preferred loss function when dealing with binary classification problems, e.g., whether a document should be classified as true or fake news.
- *metrics*: The model can evaluate a number of different metrics during training and testing. We looked at the model's accuracy, i.e., its ability to correctly predict labels for the test data in our initial fitting.

Following compilation of the model, we fit it on the training data and used that model along with the validation data to tune the parameters.

Training LSTM Model

We used the tokenized (sequential integer representations) text for our model. Given that the dimensions of the input to a LSTM model should be equal, we padded the sequences to

ensure they were uniform length. The longest sequence was 4,875 words, the shortest was 3 words, and the median was 91 words. We wanted a tradeoff between preserving as much data as possible without dimensions that were too unwieldy for training. A histogram showed that the majority of sequences were 500 words or less, so we set sequence length to 500. For sequences shorter than 500 words, we padded the beginning to ensure that the model was processing the most relevant words last. We were concerned that with post-padding, the model would partially forget the important features in a particularly short sequence as it ran through the uninformative padding. For sequences longer than 500 words, we truncated the end because we determined that the most relevant information in a news article tends to appear at the beginning.



After the text was tokenized and padded, it needed to be transformed into a word embedding. Word embeddings involve representing text with dense vectors; specifically, one can think of a word vector as the projection of a word into a continuous vector space. Given that the meaning of a word in a text depends at least in part on the words surrounding it, the position of a word vector in the vector space is learned using its proximity to other words in the text. We embedded the words in the first layer of the LSTM model, rather than outside the model.

Hypertuning Parameters for the LSTM Model

When we initially trained the LSTM model, we did not receive satisfactory scores, and so we focused our attention on tuning the hyperparameters. Initially, we tweaked the batch size (the number of training samples used during one iteration of the model) and epoch size (the number of times to pass through the training data when fitting the model) to see if that would help. However, it proved to be rather tedious, especially considering all the other parameters

that also likely needed hypertuning, so we turned to grid search. We had previously hesitated against using grid search in a deep learning environment because we were concerned about efficiency. However, given that our data set is not too large, we decided that it made sense to try grid search to improve the model's accuracy.

We tuned the following parameters to increase the LSTM model's performance:

Parameter	**Description**	**Values Tested**
drop	Dropout rate, or the fraction of the input units to set to 0 in the dropout layers	[0.2, 0.3, 0.4]
lstm_dim	Output dimension after data was processed through the LSTM layer	[16, 32, 64]
n_batch	Number of training samples used during one iteration of the model	[8, 16, 32, 64]
n_epoch	Number of times to pass through the training data when fitting the model	200 using early stopping

We initially tried to test all possible configurations of parameters, which proved to be too computationally expensive. As such, we tuned the model in segments, starting with the dropout rate and LSTM output dimension, setting the initial batch size to 32 and number of epochs to 10.

We wrote the following functions to hypertune parameters for the LSTM model:

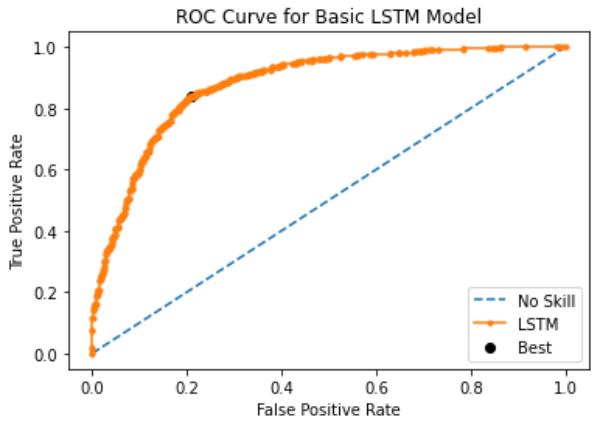
Function	**Description**
lstm_model_configs	Returns a list of all possible configurations of the parameters tested
lstm_model_fit	Creates, compiles, and fits the LSTM model using one of the parameter configurations, then returns the model
lstm_model_predict	Returns predictions on the validation data using the trained model
lstm_model_acc	Returns the accuracy score of the trained model's predictions
lstm_grid_search	Runs through each parameter configuration and returns a dictionary with the configuration as key, and the accuracy score as value

Once we had the functions written, we simply called the *lstm_grid_search* function with the list of all possible parameter configurations to find the best fit. The results showed that the best dropout rate (*drop*) was 0.3, and the best output dimension for the LSTM layer was 32.

Using the optimal *drop* and *lstm_dim* values, we reran our model to test performance by varying batch sizes. We found that the model performed best with a batch size of 64. Given that 64 was at the end of our potential values for batch size, we checked whether performance would increase with a batch size of 128, but did not see improvement. Next, we tested epoch size using our optimal parameters. In order to do so, we implemented the early stopping feature in Keras, which allows a model to train until there is no improvement, at which point it stops before all epochs have been run. We set a high epoch size (200) and the *patience* parameter to 30, meaning that the model would stop running after it had not seen any improvement in validation loss for 30 epochs. The model showed the best performance after five epochs.

Parameter	**Best Value**
drop	0.3
lstm_dim	32
n_batch	64
n_epoch	5

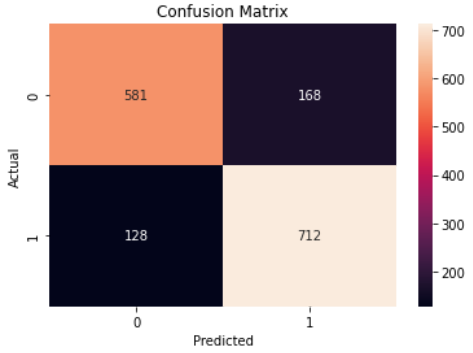
Finally, we tested if we could increase performance by adjusting the decision threshold. Specifically, we used a receiver operating characteristic (ROC) curve to check performance at various decision thresholds. A ROC curve plots the true positive rate (also known as sensitivity or recall), which is a model's ability to find all instances of the



positive class, against its false positive rate. Using our model, we found that the best threshold for predicting classes was 0.67. The area under the ROC curve–how well the model distinguishes between the classes–was 0.88. Based on the ROC curve chart, the LSTM model (displayed in orange) certainly performs better than using a model with no skill (blue dashed line).

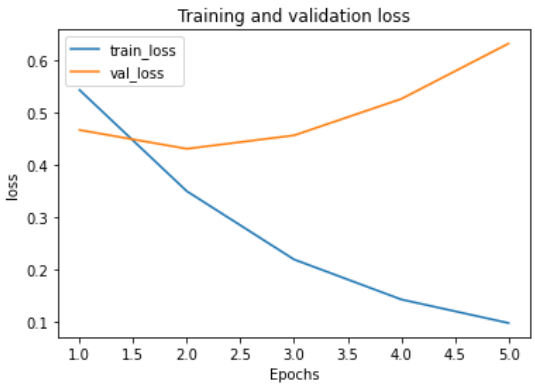
Results from the Basic LSTM Model

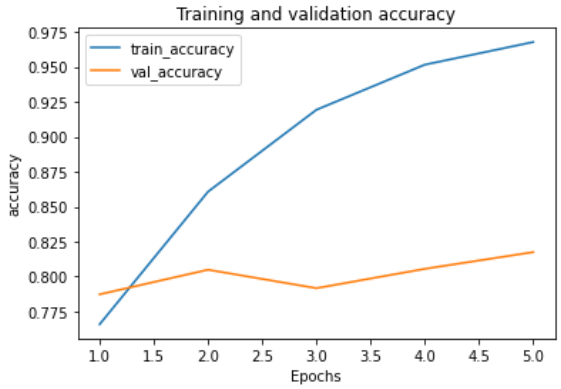
We used our optimal parameters and decision threshold to test the model's performance on the test dataset. As detailed in the below table, we obtained an overall accuracy of 0.81, with an F1-score of 0.80 for samples labeled 0, and 0.83 for those labeled 1, suggesting that the model does a better job of correctly labeling true news.



	Precision	**Recall**	**F1-Score**	**Support**	**Accuracy**
Label = 0	0.82	0.78	0.80	749	
Label = 1	0.81	0.85	0.83	840	
Total					0.81

We reviewed the training and validation loss for the basic LSTM model over the model's training. Based on the charts below, the model appears to be overfitting. The model quickly approaches high training accuracy (almost 97.5% at five epochs) and low training loss (around 10% at five epochs). The results for the training data are in stark contrast to the test data, which shows little improvement over the course of training. Going forward, more resources should be spent reviewing this discrepancy and determining what steps can be taken to reduce overfitting while increasing the model's performance on test data.

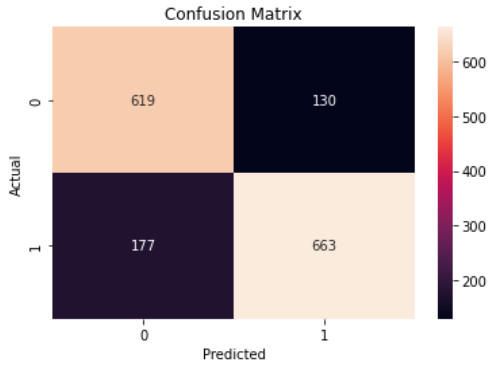




Alternate Strategies for the LSTM Model

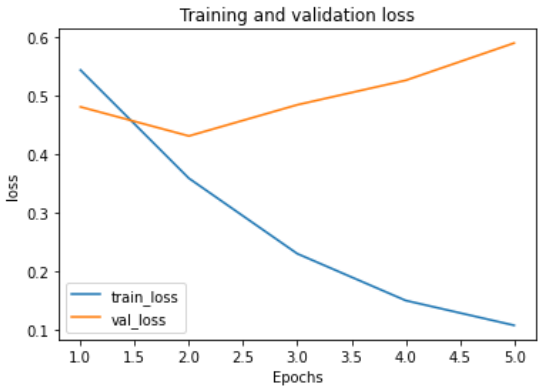
We tested variations to the basic LSTM model to see if we could increase performance. If we had more time and resources, we would try additional strategies, including the addition of a CNN layer, as well as performing hyperparameter tuning for each variation to see if different parameters would perform better based upon that particular LSTM model.

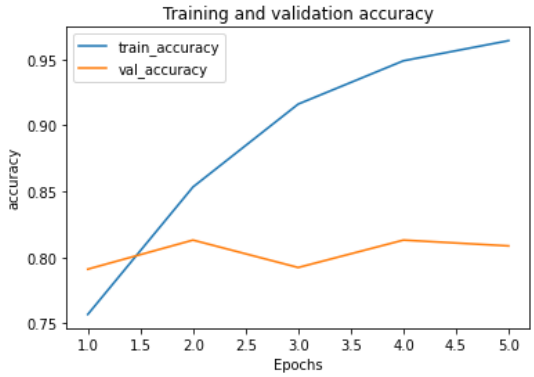
Bidirectional LSTM Model. We tested a model identical to the basic LSTM, except that we used a bidirectional LSTM layer in place of the LSTM layer. A bidirectional LSTM consists of two LSTMs: One moves the inputs through the model in a forward direction, and the other moves the inputs through the model in a backwards direction. Unfortunately, this did not increase performance (see table below). The F1-score for samples labeled 0 stayed the same, but the score for samples labeled 1 decreased to 0.81. We expected to achieve better results than this.



	Precision	Recall	F1-Score	Support	Accuracy
Label = 0	0.78	0.83	0.80	749	
Label = 1	0.84	0.79	0.81	840	
Total					0.81

The training and validation loss and accuracy for the bidirectional LSTM look similar to those for the basic LSTM model. The model quickly achieved high accuracy and low loss within just five epochs of the model, with little improvement in the validation data. We expected to see better results for the bidirectional LSTM than the basic LSTM, but that was not the case.





LSTM Model with Additional Dense Layer. We also tested a model that added an additional dense layer and dropout layer before the final dense layer that output the predictions. Unfortunately, per the table of results below, this model achieved lower scores than the previous LSTM models we tested.

	Precision	**Recall**	**F1-Score**	**Support**	**Accuracy**
Label = 0	0.77	0.80	0.79	749	
Label = 1	0.82	0.79	0.80	840	
Total					0.79

Bidirectional Encoder Representations from Transformers (BERT)

Given the unsatisfactory results from the LSTM model, we turned to BERT. BERT, which arose from a paper published by researchers at Google AI Language, works as follows:

> BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create

> state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications. (Devlin et al., 2018, p. 1).

Hypertuning Parameters for the BERT Model

Even without being fine tuned, the BERT model achieved our best accuracy rate of 89%. Initially, we tuned the model manually, stratifying training and validation data to account for class imbalance. Thereafter, we used the huggingface library to tune the following parameters:

Parameter	**Description**	**Best Value**
lr	The learning rate, or amount to adjust our model in response to the loss function	1e-5
layer_norm_eps	Epsilon, or a very small number to prevent division by 0 during convergence of the model	1e-8
epoch	Number of iterations over the data to train the model	2

Results from the BERT Model

Our BERT model returned accuracy and F1-scores of 89%, the best to date.

Robustly Optimized BERT Pretraining Approach (RoBERTa)

The final model that we tested was RoBERTa, which is a variation of the BERT model. RoBERTa makes the following modifications to BERT: "(1) training the model longer, with bigger batches, over more data; (2) removing the next sentence prediction objective; (3) training on longer sequences; and (4) dynamically changing the masking pattern applied to the training data" (Liu et al., 2019, p. 10). Specifically, RoBERTa not only improves upon the manner in which BERT was trained, but also uses significantly more data (a "new dataset (CC-NEWS) of comparable size to other privately used datasets, to better control for training set size effects") and computing power. (Liu et al., 2019, p. 1).

Training the RoBERTa Model

We trained the RoBERTa model using automated machine learning (AutoML), which automates all aspects of the process of training a machine learning model, from processing of the data to hypertuning the model's parameters, to deploying the model. Given the fact that we used AutoML for this portion of our project, we did not engage in hypertuning any parameters.

Results from the RoBERTa Model

The RoBERTa model achieved the best results of all the models that we tested:

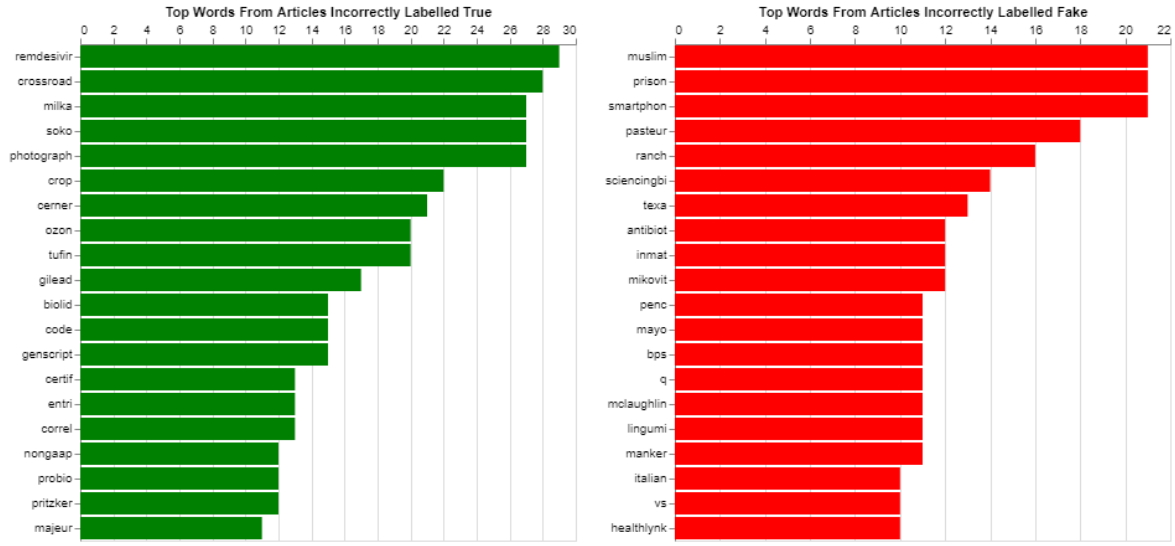
Metric	**Score**
Accuracy	0.95
Precision	0.95
Recall	0.96
AUC	0.99
F1-score	0.95

We believe that the accuracy score for classification of COVID-19 misinformation increased under this model because the data introduced to RoBERTa was more relevant to the news articles.

Moving Forward

The most limiting part of the analysis was the quality of the input data. While we spent a significant amount of time and money gathering and scraping the data to make sure our true and false data were equally distributed across time and keywords, there is still cleaning that needs to be done. We would love to try to first correct misspelling before running our models. We also hope to try removing proper nouns to see if that could increase the accuracy of the model as well.

In order to further validate our dataset, we plan to perform more robust failure analysis moving forward. The graph below shows which words appeared most commonly in incorrectly labelled data:



We hope to further expand on this as many forms of data validation have a “start at the end” approach as you see above.

Secondly, we would like to make more modifications to the LSTM model. With more resources, we could test many more variations in the layers included as well as the various parameters. We would also be interested in performance by removing more of the short articles and using the length of the longest sequence to pad all other sequences.

Along with more intense hypertuning of the LSTM model, we would also like to try a Convolutional Neural Network to see if this type of deep learning model can produce better results. If we add any images found in the articles, we have the potential to create a more robust and useful model. This can be done by using a CNN-LSTM hybrid.

Finally, we soon hope to deploy a workable model for our classification system to the public. We would like to build a platform which users can access to quickly see if an article is True or Fake, and % likelihood of each prediction. After all, a Fake prediction with 51%

probability is much different than a Fake prediction with 99% probability. We would have to build a website/app for this, but also maintain it indefinitely to protect from misuse and concept drift - no small task.

Conclusion

In this paper, we have presented our approach to effectively classify social media posts and news articles about COVID-19 as either True or Fake using a variety of machine learning and deep learning methods. Our method differs from current literature in its multiple approaches, and dataset which includes the most current news sources (through December 2021). To create novel data, we not only used reputable pre-labeled datasets mentioned above, we created new datasets from scraped news articles that include a variety of reputable and non-reputable websites, which we manually labelled. Given this new dataset, we created our binary classifiers – logistic regression, random forest, gradient boosting machine, ensemble method, long short-term memory, BERT, and RoBERTa. Experimental testing of our models concluded that our top model, RoBERTa, was able to correctly classify news articles about COVID-19 as being True or False 95% of the time, followed by BERT, which reached an accuracy of 89%. Moving forward, we hope to improve the integrity of our dataset, test new models, and deploy a finished product to the public in an ethical and sustainable manner.

References

Devlin, J., Chang, M, Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *Google AI Language*. https://arxiv.org/pdf/1810.04805.pdf

Koirala, A. (2021). COVID-19 fake news dataset. *Mendeley Data*, 1. https://data.mendeley.com/datasets/zwfdmp5syg/1

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). RoBERTa: A robustly optimized BERT pretraining approach. *arXivLabs*. https://arxiv.org/pdf/1907.11692.pdf

Ma, T., Zhou, H., Tian, Y., & Al-Nabhan, N. (2021). A novel rumor detection algorithm based on entity recognition, sentence reconfiguration, and ordinary differential equation network. *Neurocomputing*, *447*, 224-234. https://doi.org/10.1016/j.neucom.2021.03.055

Nasir, J. A., Khan, O. S., & Varlamis, I. (2021). Fake news detection: A hybrid CNN-RNN based deep learning approach. *International Journal of Information Management Data Insights*, *1(1)*, 100007. https://doi.org/10.1016/j.jjimei.2020.100007

Saenz, J. A., Gopal, S. R. K., & Shukla, D. (2021). Covid-19 fake news infodemic research dataset (CoVID19-FNIR Dataset). *IEEE Dataport*. https://dx.doi.org/10.21227/b5bt-5244

Shapiro, J. N., Oledan, J., & Siwakoti, S. (2020). ESOC COVID-19 misinformation dataset. *Empirical Studies of Conflict Project, Princeton University*. https://esoc.princeton.edu/publications/esoc-covid-19-misinformation-dataset

Sicilia, R., Giudice, S. L., Pei, Y., Pechenizkiy, M., & Soda, P. (2018). Twitter rumour detection in the health domain. *Expert Systems with Applications*, *110*, 33-40. https://doi.org/10.1016/j.eswa.2018.05.019

Siwakoti, S., Yadav, K., Thange, I., Bariletto, N., Zanotti, L., Ghoneim, A., & Shapiro, J. N. (2021). Localized misinformation in a global pandemic: Report on COVID-19 narratives

around the world. *Empirical Studies of Conflict Project, Princeton University*, 1-68. https://esoc.princeton.edu/publications/localized-misinformation-global-pandemic-report-covid-19-narratives-around-world

Vishwakarma, D. K., Varshney, D., & Yadav, A. (2019). Detection and veracity analysis of fake news via scrapping and authenticating the web search. *Cognitive Systems Research*, *58*, 217-229. https://doi.org/10.1016/j.cogsys.2019.07.004

Zeng, J., Zhang, Y., & Ma, X. (2021). Fake news detection for epidemic emergencies via deep correlations between text and images. *Sustainable Cities and Society*, *66*, 102652. https://doi.org/10.1016/j.scs.2020.102652