Is This Fake News? Using Neural Networks to Detect Fake News Articles

May 6, 2021

Abstract

In recent years, misinformation and disinformation campaigns have become increasingly prevalent in the U.S. and around the world, as the ubiquity of social media has acted as a force-multiplier on their effectiveness, with unsuspecting people causing misinformation to go viral with just the click of a button. We introduce a new tool, *www.isthisfakenews.ai*, a free public web application which provides a friction-less user interface for detecting news articles with unreliable information by utilizing natural language processing and machine learning techniques. We evaluate our tool by using the accuracy metric, where we achieved 98.6% accuracy on the test dataset.

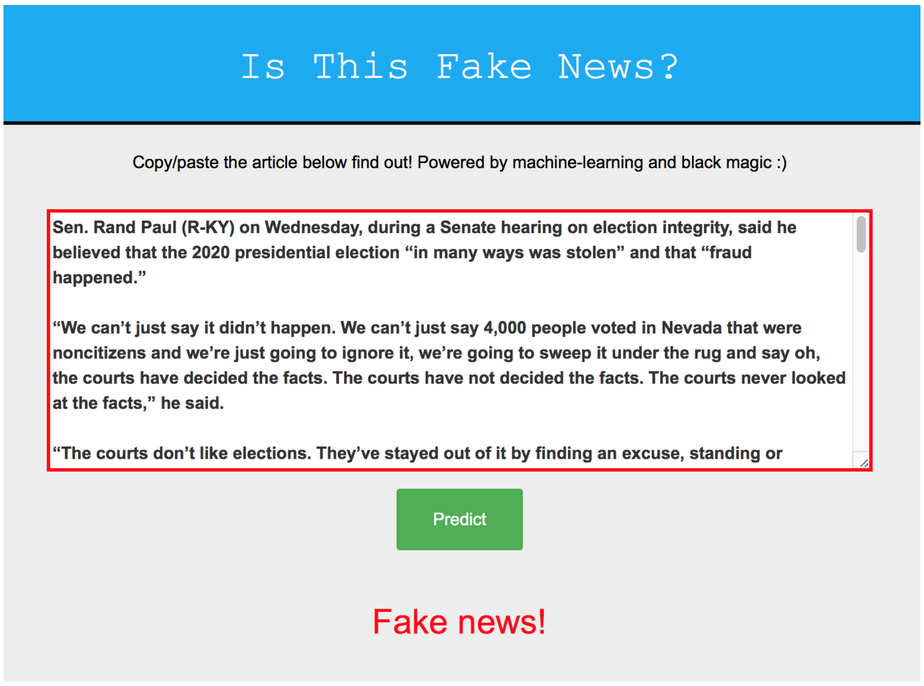
1 Introduction

As competition between nation-states shifts into the digital landscape of the 21st century, disinformation campaigns have become potent weapons for malicious actors to sow discord within their opponents' ranks. This is a new form of asymmetrical information warfare, in which a coordinated effort is made to manipulate public perceptions and inflame existing tensions between groups, in such a way that weakens the targeted nation, and is ultimately advantageous to this actor's own geopolitical interests. This disinformation, or "fake news," is manufactured to appeal to a target demographic and is distributed via social media to maximize the chances of going viral. This strategy is fairly successful; according to the U.S. Department of State, "the top one percent of fake news stories typically reached between 1,000 and 100,000 people, whereas true stories rarely exceeded an audience of one thousand." [1]

It goes without saying that verifying sources, critical analysis of information, and fact-checking are all valid strategies for defending against misinformation. These methods are effective at an individual level, but are time-consuming. In a digital culture built around convenience, we propose that at the macro-level it is more effective to automate this process to meet the following criteria:

1. Allows a user to verify information with a high degree of confidence.
2. User-experience is friction-less (i.e. only one copy/paste operation required to get a prediction).

We propose a new tool, *www.isthisfakenews.ai*, a free public web application which allows users to simply copy/paste in any article, and receive a prediction of "real news" or "fake news" in seconds. This is supported by a feed-forward multi-layer-perceptron [2], which has been trained on 30,000+ labelled examples from the "Fake and Real News" dataset on Kaggle [3].


Is This Fake News?

Copy/paste the article below find out! Powered by machine-learning and black magic :)

Sen. Rand Paul (R-KY) on Wednesday, during a Senate hearing on election integrity, said he believed that the 2020 presidential election "in many ways was stolen" and that "fraud happened."

"We can't just say it didn't happen. We can't just say 4,000 people voted in Nevada that were noncitizens and we're just going to ignore it, we're going to sweep it under the rug and say oh, the courts have decided the facts. The courts have not decided the facts. The courts never looked at the facts," he said.

"The courts don't like elections. They've stayed out of it by finding an excuse, standing or

Figure 1: Figure 1: Example output

2 Background

Attempting to identify fake news using natural language processing and machine learning is not a novel idea. However, much of the research has been focused on maximizing certain metrics and benchmarks by using sophisticated architectures, incorporating metadata with text, and more [4]. Thus, it is our contention that while the approaches are interesting from a research perspective, they are not feasible for usage by the general population in such a way that will solve this problem in the real world.

Concretely, we contend that the vast majority of people encountering a news article on Facebook and are curious about the reliability of the source/information, are simply not invested enough in finding the answer to spend time collecting the various metadata about it that these sophisticated models incorporate as features.

Instead, we hypothesize that the most practical machine learning approach for fake-news detection for real-world usage would be text-only models deployed in an application which can be used with only one copy/paste operation by user.

Therefore, in our project we focus our efforts on the finding an approach that is both practical for use in a production environment and requires minimal user interaction and/or effort to produce results that maximize usability and user engagement.

As such, some of the constraints of our problem space included:

1. Keeping the entire app and model within the 500MB memory limit to which a standard free Heroku instance is subject.
2. Creating a text-only model trained only on the article text.
3. Limiting ourselves to one copy/paste operation required by the user to input the article text.

3 Data

We used the ”Fake and real news” dataset from Kaggle [3] to train two different models and evaluate the results. This dataset contains 44,898 labelled examples of real and fake news articles. The distribution is 47.7% real and 53.3% fake, which we deemed suitable for a binary classification task.

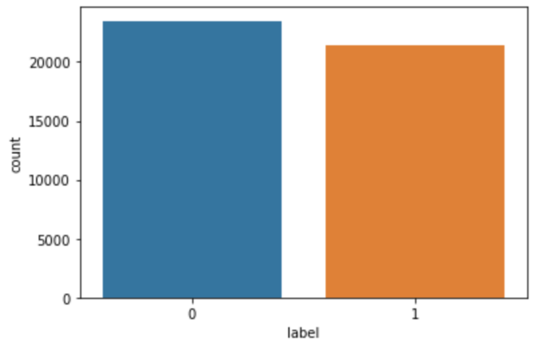
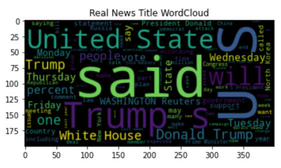


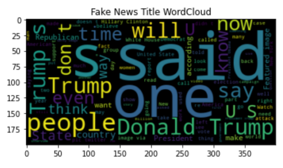
Figure 2: Figure 3: Distribution of fake vs. real news in dataset

The dataset features include: title, text, subject, date, and label. While utilizing all the features yielded slightly higher accuracy, the challenge this project attempts to solve is maximizing performance from just the article text. Therefore, we dropped all features except the article text.

We did some additional data exploration before building the models and training pipeline, such as the word clouds shown below for real and fake news.



(a) Real news word cloud



(b) Fake news word cloud

Figure 3: Figure 3: Word clouds for real vs. fake news

4 Model and Approach

We took a two-prong approach during our research and development phase, using a feed-forward multi-layer perceptron and BERT fine-tuning, and evaluating the results to determine which model to deploy to production.

4.1 Multi-Layer Perceptron

Our first approach was developing a multi-layer perceptron model training pipeline. We used code published by Mehmet Lauda Tekman [5] as a starting point and built on that to meet the requirements of this project. The primary libraries used in this approach include NLTK, SKLearn and PyTorch.

First, the data is pre-processed to replace non-alphabetic characters with whitespace, converted to lowercase, and then is tokenized by the word_tokenizer from the NLTK library. The tokens are then ”stemmed” or lemmatized by the WordNet lemmatizer from the NLTK library.

Next, feature extraction from these stemmed tokens is performed by the count vectorizer from the SKLearn library, which returns a matrix of token counts. We set the vectorizer to use only the top 4000 features/tokens, in order to keep the computational complexity reasonable.

Then, the model is defined as having an input size of 4000 (corresponding to the size of the matrices returned by the pre-processing function), 3 hidden linear layers with ReLU activation functions, and output layer of size 2.

Finally, the Adam optimizer is used with a relatively low learning rate of 0.001, and the error function used is cross entropy loss [6], since we are doing a binary classification task.

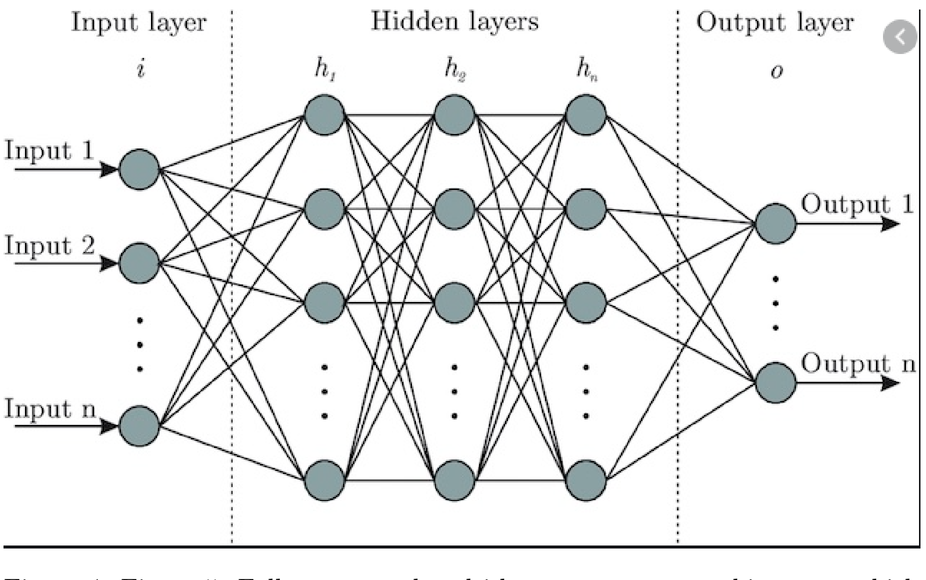


Figure 4: Figure 5: Fully-connected multi-layer perceptron architecture, which is conceptually representative of our model.

4.2 BERT Fine-tuning

Our second approach was using BERT [7] as a feature extractor by freezing the pre-trained parameters and adding a small 3 layer fully-connected classifier to the end of it, which we then trained on the dataset. We used code published by Clément Bisaillon [8] as a starting point and built on that to better fit the requirements of this project discussed above. The primary libraries used in the approach were Hugging Face and PyTorch.

First, the data is pre-processed by splitting into words and encoding it 300 words at a time through the BertTokenizer (using bert-base-uncased), which returns a list of tensors with a max size of 500.

Next, the pre-trained BertForSequenceClassification model is loaded, its parameters are frozen, and a small multi-layer perceptron is added to the end as a binary classifier which we then train.

Finally, we use a stochastic gradient descent (SGD) optimizer with a mean squared error (MSE) loss function with a slightly higher learning rate of 0.01. We then train for 1 epoch, iterating through the dataset once.

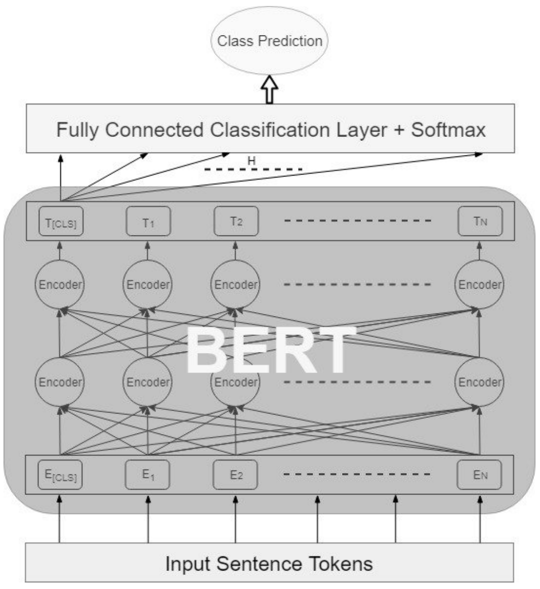


Figure 5: Figure 6: BERT fine-tuning strategy in which the outputs of BERT are fed into a fully-connected classification layer.

4.3 Software Architecture

The software architecture is fairly straight-forward, with a front-end built using plain HTML, CSS, and JavaScript, and a back-end built with the Python framework Flask (see Figure 6: architecture diagram).

The raw data is copy/pasted into the text input box by the user. Once the "predict" button is clicked, an asynchronous HTTP request is sent to the Flask '/predict' endpoint. Once the API successfully receives the article text, the back-end performs the same pre-processing steps on it as we did during model

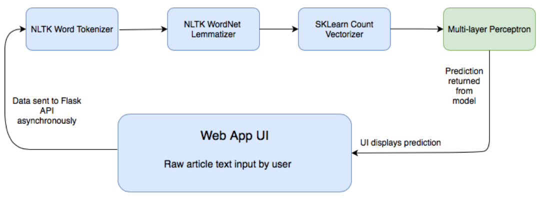


Figure 6: Figure 6: High level software architecture.

development, using the NLTK work. However, here we load into memory same count vectorizer and model parameters that were created during the training loop. This way produces a matrix dimension 4000 (representing the top 4000 most frequent tokens found in the training data) instead of creating a new one which would start counting tokens from scratch again.

We chose to use the multi-layer perceptron model in the app due to its superior performance (see evaluation section below), as well as the fact that using the BERT model caused the total build size to exceed the 500MB limit of the Heroku instance. We considered using a lighter-weight implementation of BERT, such as TinyBERT [9] and plan to explore this further, however, for the purposes of this project the multi-layer perceptron met our requirements.

One important consideration in this architecture is scalability. This is a proof-of-concept which is not scalable to potentially to millions of users. However, since the scope of this web app is very limited, and is essentially just a one-page front-end hitting an API on the back-end, there a couple ways we could scale this:

1. The model and back-end API are fundamentally stateless (i.e. if there are multiple instances of the API + model running concurrently sitting behind a load balancer, it doesn't matter which instance receives the prediction request from the front-end, the result is guaranteed to always be the consistent). Therefore, this easily can be horizontally scaled by Dockerizing the back-end Flask API and model and using Kubernetes for container orchestration to auto-scale the number of instances we need running to handle the current traffic.

2. With potentially millions of people wanting to verify a trending, popular article, it would make sense to add a caching layer (such as Redis) between the front-end and the back-end, in order to avoid having to recompute predictions for the same article more than once. This is a simple way of both optimizing performance (latency) as well as reducing load on our backend servers.

5 Evaluation

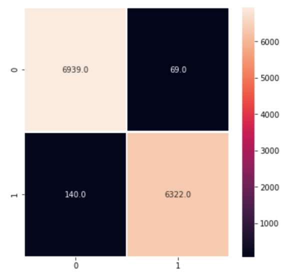
The primary metric we chose for evaluating our models' performance was accuracy; however, we also looked at the precision, recall, and F1 scores. Using the feed-forward multi-layer perceptron, we were able to achieve an accuracy of 98.6% on the test dataset, whereas the BERT fine-tuning approach achieved an accuracy of 96.7% on the test dataset. In addition, the multi-layer perceptron performed better than the BERT fine-tuning approach on all measured metrics (see Figure 7 below).

	Accuracy	Precision	Recall	F1
Multi-layer Perceptron	0.986	0.99	0.979	0.985
BERT Fine-tuning	0.967	0.96	0.972	0.966

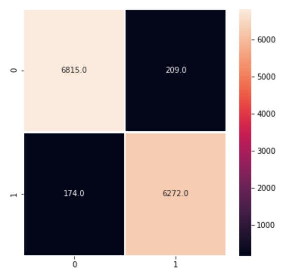
Figure 7: Figure 7: Accuracy, precision, recall, and F1 score comparison between models.

This was a somewhat surprising result, as we had initially thought that encoding the input text using a count vectorizer would be a somewhat naive approach, since this is essentially training on matrices containing the counts of the top 4000 tokens that appeared most frequently in the training data. We assumed that the BERT encodings, used as input to the fully-connected classifier layer, would contain more information for the classifier to learn from than the count vectorized tokens fed into the multi-layer perceptron.

Next, we generated confusion matrices to analyze the predictions of the models more closely.



(a) Multi-layer perceptron



(b) BERT fine-tuning

Figure 8: Figure 8: Confusion matrices for both models

Interestingly, the multi-layer perceptron had a lower rate of both false positives and false negatives.

Futhermore, the memory requirements were lower for multi-layer perceptron than the BERT fine-tuning as well. In fact, the BERT fine-tuning model did not meet the 500MB total build size constraint of the Heroku instance. In light of these findings, we determined the multi-layer perceptron model to be superior for this use case and deployed it into production.

6 Implications

With additional research and development, we believe this tool could be suitable for real-world use in fighting disinformation. The primary barrier would be reaching enough daily active users to achieve a significant dampening effect on the spread of fake news. We also think for a tool such as this to truly achieve widespread use, even navigating to the website and doing 1 copy/paste operation is too much friction. The app would probably need to integrate with browsers and apps in order for a user to simply highlight text in a tweet, Facebook post, or news article and simply "right-click" and select an option "Is this fake news?" which would query the model API and return the result.

7 Conclusion

Disinformation campaigns are a growing part of the 21st century cyber-threat landscape, and our most effective defense will be to develop tools and technologies that mitigate the effects of disinformation at a macro-level, rather than leaving it up to individuals. In this project, we proposed a new tool *isthisfakenews.ai*, a free public web app that identifies misinformation with just one copy/paste operation. During the research and development phase, we investigated two possible approaches: multi-layer perceptron and BERT fine-tuning. After evaluating the performance of the models and the requirements and constraints of the project, we chose to deploy the multi-layer perceptron model into production as it achieved a 98.6% accuracy score while maintaining a total size of less than the 500MB limit of the Heroku instance it is deployed on.

References

[1] U.S. Department of State. Weapons of mass distraction, 2019.

[2] Wikipedia. Feedforward neural network (multilayer perceptron), 2021.

[3] Clément Bisaillon. Fake and real newss dataset, 2020.

[4] William Yang Wang. "liar, liar pants on fire": A new benchmark dataset for fake news detection, 2017.

[5] Mehmet lauda Tekman. Detailed fake news classification with pytorch, 2020.

[6] Wikipedia. Cross entropy, 2021.

[7] Wikipedia. Bert (language model), 2019.

[8] Clément Bisaillon. Classifying fake news with bert, 2020.

[9] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding, 2020.