CIS 5526 final project report: Real vs. Fake News Prediction

Nick McCloskey

Introduction

While captivating but inaccurate headlines designed to sell information have always been a persistent problem in media, the advent of social media has enabled fake news to propagate with alarming rapidity, enough to cause large-scale impacts. Research conducted by Common Sense Media reveals that 31% of kids ages 10-18 have shared at least one news article later discovered to be fake (1). A large part of the problem seems to be the overwhelming quantity of information available. The Observatory on Social Media has found that information overload alone can account for the virality of fake news, because as the number of memes at play increases, the information quality diminishes due to limited attentional capacity (2).

Whatever its etiology, misinformation can have serious consequences. Consider for example the "Pizzagate" incident, in which Edgar Maddison Welch fired an AR-15 assault rifle inside a Comet Ping Pong as a vigilante effort to rescue the children he believed were being trafficked from the ping pong bar/pizzeria. This belief came from pre-election conspiracy theories circulating on the internet. Such fake election news was found to outperform real news in terms of online engagement (3). Because detecting misinformation is an important capacity in the face of misinformation and information overload, building a fake news predictor is a relevant and potentially crisis-averting natural language processing (NLP) problem. These predictors could help filter the content on media platforms so that their users could be more confident in the veracity of the information they encounter there. Cleaner information would also decrease the likelihood of another Pizzagate. Fortunately, researchers have already made significant progress addressing this issue with machine learning techniques. Neural network and support vector machine approaches to fake news detection have reached accuracies of 99.9% using a hybrid text classification method (4). A logistic classifier trained on the text from the title and post has achieved 99.4% accuracy (5).

Approach

The overall purpose of this NLP project was to develop and compare machine learning models that predict if a news article is fake or real from its title and content. The goals were to use three datasets of real and fake news articles (Part 1) to train long short-term memory (LSTM) neural networks, (Part 2) conduct a comparative analysis of two vectorizers and several other machine learning algorithms, and (Parts 1 and 2) investigate how well all these models perform when tested on similarly preprocessed data from the other datasets to model the situation where only one dataset was available for model training before encountering the other examples. Because the LSTM models involved tokenizer preprocessing and much longer training time, those comparisons were conducted separately. Hours working on this project were tracked closely in memory until hour 20, at which point there was an intention to limit the remaining work to 5 hours. This is explained to clarify that the following are estimations: 2-3 hours EDA, 4-5 hours part 1, 5-6 hours part 2, 4-5 hours revamping parts 1 and 2 (more streamlined code, another dataset), 2 hours debugging, 1 hour figure generation, 4-6 hours writing and editing the report.

NLP connects human language with computer language using linguistics and mathematics. For parsing text or speech into computer-compatible data, formal (or prescriptivist) linguistics that codifies the abstract rules of a language is less useful than its computational (or statistical, or descriptive) counterpart that identifies patterns in actual linguistic data due to the continually evolving, error-prone, and inconsistent nature of language. Various preprocessing methods have been invented to prepare sequences of text for machine learning, and those employed in this project are white-space tokenization, bag of words representation (by count vectorization), and term-frequency-inverse document frequency (TF-IDF) vectorization.

<u>Table 1: Dataset attributes:</u> Counts of examples, number of unique tokens and maximum token length in dataset.

	Dataset 1	Dataset 2	Dataset 3
fake examples	23481	3164	36509
real examples	21417	3171	35028
total examples	44898	6335	71537
unique words	114617	65307	231378
max length	4596	8710	18446

The datasets as well as many helpful Python code examples come from the Kaggle database (6,7,8). All datasets are CSV formatted and contain columns for the article title and content, each with roughly even numbers of articles labeled as real and fake (Table 1). The first steps of preprocessing entailed dropping rows with missing values, combining the title and text columns, extracting tokens, removing stop words, joining all tokens into a string, and shuffling the data, which generally helps neural networks avoid getting stuck in a local minimum. Tokenization breaks the text stream into meaningful units such as symbols, words, sub-words, or sentences to ultimately produce a numerical vector of token frequencies. Many libraries exist for tokenization; this project applies one from Gensim to generate lists of meaningful tokens, and another from Tensorflow for the LSTM model because it can fit itself on a vocabulary of training data and convert that data to vectors of token indices. The Gensim tokenizer by default splits a text sequence at the white spaces, converts all characters to lower case, and keeps only words with lengths between three and fifteen characters in the output list. Stop words are words such as "are", "such", and "as" that allow the sentence to function in its human context but do not provide meaningful information to the computer. Their removal reduces noise and dimensionality. Combining title with text collects all available information into one vector, and combining tokens into strings following the examples in (7,9) results in input compatible with final preprocessing steps. Another important preprocessing step is padding shorter sequences with empty values because documents such as these will have varying word lengths, but LSTM neural networks require inputs of the same size (10). Here, padding was added to the end of sequences.

Part 1 of the project trained three models of type LSTM, a recurrent neural network superior to the traditional ones, selected for its ability to "memorize" information that enables a more meaning-based prediction than other models which parse each words separately into their own categories. This kind of processing is better aligned with human language perception and is made possible by three main logic gates (the forget, input, and output gates) which retain useful information and dispense with the irrelevant (11). Based on the example in (7), each dataset was used to fit a sequential model that began with an embedding layer to facilitate learning from large inputs (10), followed by a bidirectional layer to allow input to flow in both directions and preserve future and past information (12), followed by two dense layers with Relu activation because this transfer function is computationally efficient and entails no vanishing gradient problem, and finally ending with a single-neuron output layer with sigmoid activation – a common transfer function for the output in binary classification tasks. In model compilation, the Adam optimizer, an extension of stochastic gradient descent which includes momentum (remembering the general direction of the gradient), is selected because it requires fewer epochs to converge.

The three datasets were partitioned into 75% training and 25% testing data, and the Tensorflow tokenizer was fit on the training portion using the number of unique words in the dataset (which became the number of inputs to each neuron in the first layer). Padding was established based on the maximum length sequence in the dataset. The vectors of token indices from each training partition fit three LSTM models for only one epoch (because of long training time) with a batch size of 64 and a validation split of 10%. The testing portions of each dataset were preprocessed with the tokenizer and padder of the dataset on which the model was fit, and used to test the accuracy of the model, generating a total of 9 scores.

Part 2 used two types of vectorizers to preprocess data: count and TF-IDF. The former is an implementation of the bag-of-words model and produces a vector of token counts to represent the frequency of each word in the entire text corpus (here, Gensim-preprocessed strings joining tokens of titles and articles). TF-IDF, however, incorporates information about token significance, which allows it to remove those less important for analysis, simplifying the model by reducing input dimensions (13,14). This vectorizer produces token values that increase with their frequency in the document, but that are offset by the number of documents containing them, thus correcting for generally frequent and document non-specific words (15). The maximum features parameter, which determines the number of most frequent tokens to include, was set to 1000 for each vectorizer to make training computationally feasible (16). After being preprocessed as described above, the datasets were partitioned (75/25 train/test split), vectorized, and used to train each of five scikit-learn classifiers: Random Forests (RF), Gaussian Naïve Bayes (GNB), Linear Support Vector Classifier (SVC), k-Nearest Neighbors (kNN), and Logistic Regression (LR).

Decision trees use a branching structure to evaluate criteria along a series of nodes to make a prediction at the "leaves" or tips of the hierarchy. The RF classifier represents an ensemble of such trees trained on either the entire dataset or various subsets thereof whose averaged prediction achieves an accuracy greater than any individual model could. Hyperparameters include number of trees in the forest (here, 100) and maximum examples for training each tree (none set, so the entire dataset was used) (18). A GNB classifier is based on Bayes' theorem, and it assumes conditional independence between all features given the target label (hence naïve) and that the likelihood of the features comes from a Gaussian distribution. In binary classification, predictions are made based on the relative probabilities of observing the features given either label (19). SVC involves finding a hyperplane to separate data points in a space with as many dimensions as there are features, maximizing the marginal distance between the support vectors (those closest to the hyperplane), and predicting label based on which side of the hyperplane data lie on (20). The kNN algorithm uses "guilt by association" logic to predict the class of a new data point from the labels of those data points closest in feature space. Hyperparameters include number of neighbors to consult (here, 7) and how to weight their input in the prediction (here, uniform) (21,22). Finally, the LR classifier employs linear regression to predict the probability that a given data point belongs to a category and produces binary output (23). Testing portions of each dataset were fit on the vectorizer from the training data for each model and used to test accuracy, resulting in a total of (3x2x5x3) 90 scores.

Results

First, exploratory data analysis is presented for Dataset 1 (D1), which also contains columns for article subject and date. For Figure 1 (see appendix for Figures 1-5), articles in D1 were grouped by subject and totaled for the real and fake news categories. There is a prominent imbalance in the distribution of fake and real news articles across subjects, with real examples falling into only two categories (politics and world news), while all categories are represented with at least a few hundred fake news examples. All words from each of the real and fake news categories of examples were concatenated into a large string to generate word clouds. Figure 2 displays the most frequent (largest) tokens present in the real and fake news pools in D1. Standing out in both figures unsurprisingly are the tokens "united state", "white house", and "donald trump." In the real news pool, "north korea" is relatively large and "hillary clinton" small, while in the fake news data "hillary clinton" ends up second only to "donald trump" (not for the first time). This latter observation suggests Clinton enjoys a considerable representation in fake news, and tracks with the "Pizzagate" example above, as the aggressor believed her to be behind the Comet Ping Pong criminal activity (3). The ex-president is prevalent throughout either pool of examples. Figure 3 displays a histogram of word counts across fake and real news examples. The counts range from 3 to 4,596 words (hence the zoomed in subfigure), but most articles are less than 400 words long. It appears from the distinct distribution shapes that fake news articles tend to be shorter.

For Part 1, each LSTM model was tested on each dataset, and the accuracy table is shown in Figure 4. The models trained on Datasets 1 and 2 performed best on test data from those respective datasets, as expected. The third model, however, performed better predicting data from D1 than from D3; it also outperformed the D2 model on the D2 test data. As the D1-D1 (training-testing) accuracy was highest overall, it seems that D1 contains data most amenable to accurate prediction. Perhaps this is due to a factor such as a salient prevalence of one token in one type of article, such as "hillary clinton" discussed above. But this does not seem to result in the most robust model, as the D1-D2 accuracy (49.8%) is similar to what would be achieved by random guessing, and the D1-D3 accuracy, while better (81.4%) is lackluster compared to D1-D1 (99.9%).

For Part 2, the accuracy scores of the comparative study of five non-neural network classifiers and two vectorizers are displayed in Figure 5. In the y-axis labels, the first number indicates the dataset used to fit the vectorizer and train the model, the letter indicates the count (c) or TF-IDF (t) vectorizer applied to both training and testing data, and the second number indicates the dataset from which the test partition comes. Generally, the dataset performance patterns are consistent with those observed in Part 1, with accuracies highest on D1 testing data, lowest on D2, and most consistent from D3-trained models. The D1-D1 and D1-D3 Random Forest model scores are highest across both vectorizers (all ~%99.8%), and RF models achieve the highest overall accuracy (mean 83.32%), followed by Logistic Regression (82.32%) and linear SVC (81.66%). The k-Nearest Neighbors and Gaussian Naïve Bayes models perform worse, with average accuracies of 75.40% and 73.36% respectively. For some tests, (e.g., D1-D3-GNB) the TF-IDF vectorizer results in a modest (~4%) accuracy increase, as would be expected from the claim of its superiority to count vectorization in (14) due to its including more information about word significance. For other tests, however, (D3-D2-GNB) the count vectorizer achieves a ~10% greater accuracy than TF-IDF. Considering the averaged scores of the count (78.93%) and the TF-IDF (79.49%) vectorizers, it appears that perhaps the latter is generally better, but only very slightly, and not in all cases.

Conclusion

The prevalence and virulence of fake news has resulted in violence, so the ability to distinguish truth from falsehood in the media is important. Robust classifiers represent a potentially effective filter that could be incorporated to limit the spread of fake news stories across various media platforms. This project investigated the machine learning problem of using NLP to develop models that predict if a news article is real or fake from its title and content. Three datasets of different sizes, two vectorizers, and six total ML algorithms were implemented for comparison.

In Part 1, the smallest dataset results in poorest overall LSTM model performance, the largest dataset results in best overall performance, and the medium dataset offers the seemingly greatest opportunity (seen in D1-D1 and D3-D1, but not D2-D1 scores) for high accuracy. These results are consistent with the fact that training neural networks on large datasets helps combat the problems of high variance and overfitting (24) and indicate that predictability is not directly related to dataset size. In Part 2, the Random Forest, linear SVC, and Logistic Regression models all performed generally well, but RF trained on D3 achieved the most consistently high accuracy scores. The use of either the count or TF-IDF vectorizer in some cases made a notable difference, but not always in the expected direction, and the general (average) difference is marginal. As in Part 1, the robustness of D3-trained models and high accuracy of D1-tested models was observed.

It is acknowledged that model performance can vary greatly with different choices of hyperparameters to govern the learning process, so all results and conclusions may not reflect the models generally. Investigating the accuracies resulting from making systematic adjustments to model hyperparameters would be an interesting next step. Overall, the results support the findings (e.g., 6, 25) that LSTM and RF models are particularly effective predictors of fake news, and robust models (trained on large quantities of data) are applicable – sometimes with greater success – to other datasets.

References

1. https://reader.elsevier.com/reader/sd/pii/S1877050917323086?token=B92A0B814B73FF22BDB3CB2520C78F779970379DDB92DFE48376406C03566AD5FF97B30620B0D970FAF5BBB09B1A4552&originRegion=us-east-1&originCreation=20221108160546
2. https://www.scientificamerican.com/article/information-overload-helps-fake-news-spread-and-social-media-knows-it/
3. https://www.pbs.org/newshour/science/real-consequences-fake-news-stories-brain-cant-ignore
4. https://iopscience.iop.org/article/10.1088/1757-899X/1099/1/012040/pdf
5. https://www.sciencedirect.com/science/article/pii/S1877050918318210
6. https://www.kaggle.com/datasets/clmentbisaillon/fake-and-real-news-dataset
7. https://www.kaggle.com/code/sanchukanirupama/lstm-based-fake-news-detection
8. https://www.kaggle.com/datasets/saurabhshahane/fake-news-classification
9. https://www.kaggle.com/code/paramarthasengupta/fake-news-detector-eda-prediction-99
10. https://medium.com/@canerkilinc/padding-for-nlp-7dd8598c916a
11. https://www.analyticsvidhya.com/blog/2021/06/lstm-for-text-classification/
12. https://developers.google.com/machine-learning/crash-course/embeddings/video-lecture
13. https://analyticsindiamag.com/complete-guide-to-bidirectional-lstm-with-python-codes/
14. https://towardsdatascience.com/basics-of-countvectorizer-e26677900f9c
15. https://medium.com/analytics-vidhya/fundamentals-of-bag-of-words-and-tf-idf-9846d301ff22
16. https://stackoverflow.com/questions/46118910/scikit-learn-vectorizer-max-features
17. https://www.linkedin.com/pulse/count-vectorizers-vs-tfidf-natural-language-processing-sheel-saket#:~:text=TF%2DIDF%20is%20better%20than,by%20reducing%20the%20input%20dimensions.
18. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
19. https://iq.opengenus.org/gaussian-naive-bayes/
20. https://www.kaggle.com/code/xingewang/the-math-behind-linear-svc-classifier
21. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
22. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
23. https://www.geeksforgeeks.org/understanding-logistic-regression/
24. https://www.analyticsvidhya.com/blog/2021/10/ensemble-modeling-for-neural-networks-using-large-datasets-simplified/#:~:text=Training%20on%20large%20datasets%20helps,number%20of%20samples%20of%20data.
25. https://www.kaggle.com/code/vangelistsiatouras/fake-news-detection-random-forests-99-9-acc