Fake News Detection

John Curci
College of Engineering
Boston University
Boston, MA 02215
jcurci92@bu.edu

Keval Khara
College of Engineering
Boston University
Boston, MA 02215
kevalk@bu.edu

Ashwin Pillai
Metropolitan College
Boston University
Boston, MA 02215
ashwin96@bu.edu

Ruoxi Qin
College of Arts and Sciences
Boston University
Boston, MA 02215
rxqin@bu.edu

Abstract

In this paper, we explore the application of Natural Language Processing techniques to identify when a news source may be producing fake news. We use a corpus of labeled real and fake new articles to build a classifier that can make decisions about information based on the content from the corpus. We use a text classification approach, using four different classification models, and analyze the results. The best performing model was the LSTM implementation, which f

The model focuses on identifying fake news sources, based on multiple articles originating from a source. Once a source is labeled as a producer of fake news, we can predict with high confidence that any future articles from that source will also be fake news. Focusing on sources widens our article misclassification tolerance, because we then have multiple data points coming from each source.

1 Introduction

Fake news, defined as a made-up story with an intention to deceive, has been widely cited as a contributing factor to the outcome of the 2016 United States presidential election. While Mark Zuckerberg, Facebook's CEO, made a public statement [1] denying that Facebook had an effect on the outcome of the election, Facebook and other online media outlets have begun to develop strategies for identifying fake news and mitigating its spread. Zuckerberg admitted identifying fake news is difficult, writing, ”This is an area where I believe we must proceed very carefully though. Identifying the truth is complicated.”

Fake news is increasingly becoming a menace to our society. It is typically generated for commercial interests to attract viewers and collect advertising revenue. However, people and groups with potentially malicious agendas have been known to initiate fake news in order to influence events and policies around the world. It is also believed that circulation of fake news had material impact on the outcome of the 2016 US Presidential Election [2].

2 Data

The datasets used for this project were drawn from Kaggle [3]. The training dataset has about 16600 rows of data from various articles on the internet. We had to do quite a bit of pre-processing of the data, as is evident from our source code [4], in order to train our models.

A full training dataset has the following attributes:

1. id: unique id for a news article
2. title: the title of a news article
3. author: author of the news article
4. text: the text of the article; incomplete in some cases
5. label: a label that marks the article as potentially unreliable
    - 1: unreliable
    - 0: reliable

3 Feature Extraction and Pre-Processing

The embeddings used for the majority of our modelling are generated using the Doc2Vec model. The goal is to produce a vector representation of each article. Before applying Doc2Vec, we perform some basic pre-processing of the data. This includes removing stopwords, deleting special characters and punctuation, and converting all text to lowercase. This produces a comma-separated list of words, which can be input into the Doc2Vec algorithm to produce an 300-length embedding vector for each article.

Doc2Vec is a model developed in 2014 based on the existing Word2Vec model, which generates vector representations for words [5]. Word2Vec represents documents by combining the vectors of the individual words, but in doing so it loses all word order information. Doc2Vec expands on Word2Vec by adding a ”document vector” to the output representation, which contains some information about the document as a whole, and allows the model to learn some information about word order. Preservation of word order information makes Doc2Vec useful for our application, as we are aiming to detect subtle differences between text documents.

4 Models

4.1 Naive Bayes

In order to get a baseline accuracy rate for our data, we implemented a Naive Bayes classifier. Specifically, we used the scikit-learn implementation of Gaussian Naive Bayes. This is one of the simplest approaches to classification, in which a probabilistic approach is used, with the assumption that all features are conditionally independent given the class label. As with the other models, we used the Doc2Vec embeddings described above. The Naive Bayes Rule is based on the Bayes' theorem

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)} \tag{1}$$

Parameter estimation for naive Bayes models uses the method of maximum likelihood. The advantage here is that it requires only a small amount of training data to estimate the parameters.

4.2 Support Vector Machine

The original Support Vector Machine (SVM) was proposed by Vladimir N. Vapnik and Alexey Ya. Chervonenkis in 1963. But that model can only do linear classification so it doesn't suit for most of the practical problems. Later in 1992, Bernhard E. Boser, Isabelle M. Guyon and Vladimir N. Vapnik introduced the kernel trick which enables the SVM for non-linear classification. That makes the SVM much powerful.

We use the Radial Basis Function kernel in our project. The reason we use this kernel is that two Doc2Vec feature vectors will be close to each other if their corresponding documents are similar, so the distance computed by the kernel function should still represent the original distance. Since the Radial Basis Function is

$$K(x, x') = \exp\left(-\frac{||x - x'||^2}{2\sigma^2}\right) \tag{2}$$

It correctly represents the relationship we desire and it is a common kernel for SVM.

We use the theory introduced in [6] to implement the SVM. The main idea of the SVM is to separate different classes of data by the widest "street". This goal can be represented as the optimization problem

$$\arg\max_{w,b}\left\{\frac{1}{||w||}\min_{n}\left[t_n(w^T\varphi(x_n) + b)\right]\right\} \tag{3}$$

$$s.t. \quad t_n(w^T\varphi(x_n) + b) \geq 1, \qquad n = 1, \ldots N \tag{4}$$

Then we use the Lagrangian function to get rid of the constraints.

$$L(w, b, a) = \frac{1}{2}||w||^2 - \sum_{n=1}^{N} a_n \left\{t_n(w^T\varphi(x_n) + b) - 1\right\} \tag{5}$$

where $a_n \geq 0, n = 1, \ldots N$.

Finally we solve this optimization problem using the convex optimization tools provided by Python package CVXOPT.

4.3 Feed-forward Neural Network

We implemented two feed-forward neural network models, one using Tensorflow and one using Keras. Neural networks are commonly used in modern NLP applications [7], in contrast to older approaches which primarily focused on linear models such as SVM's and logistic regression. Our neural network implementations use three hidden layers. In the Tensorflow implementation, all layers had 300 neurons each, and in the Keras implementation we used layers of size 256, 256, and 80, interspersed with dropout layers to avoid overfitting. For our activation function, we chose the Rectified Linear Unit (ReLU), which has been found to perform well in NLP applications [7].

This has a fixed-size input $x \in \mathbf{R}^{1\times300}$

$$h_1 = \text{ReLU}(W_1x + b_1) \tag{6}$$

$$h_2 = \text{ReLU}(W_2h_1 + b_2) \tag{7}$$

$$y = \text{Logits}(W_3h_2 + b_3) \tag{8}$$

4.4 Long Short-Term Memory

The Long-Short Term Memory (LSTM) unit was proposed by Hochreiter and Schmidhuber[8]. It is good at classifying serialized objects because it will selectively memorize the previous input and use that, together with the current input, to make prediction. The news content (text) in our problem is inherently serialized. The order of the words carries the important information of the sentence. So the LSTM model suits for our problem.

Since the order of the words is important for the LSTM unit, we cannot use the Doc2Vec for pre-processing because it will transfer the entire document into one vector and lose the order information. To prevent that, we use the word embedding instead.

We first clean the text data by removing all characters which are not letters nor numbers. Then we count the frequency of each word appeared in our training dataset to find 5000 most common words and give each one an unique integer ID. For example, the most common word will have ID 0, and the second most common one will have 1, etc. After that we replace each common word with its assigned ID and delete all uncommon words. Notice that the 5000 most common words cover the most of the text, as shown in Figure 1, so we only lose little information but transfer the string to a list of integers. Since the LSTM unit requires a fixed input vector length, we truncate the list longer than 500 numbers because more than half of the news is longer than 500 words as shown in Figure 2. Then for those list shorter than 500 words, we pad 0's at the beginning of the list. We also delete the data with only a few words since they don't carry enough information for training. By doing this, we transfer the original text string to a fixed length integer vector while preserving the words order information. Finally we use word embedding to transfer each word ID to a 32-dimension vector. The word embedding will train each word vector based on word similarity. If two words frequently appear together in the text, they are thought to be more similar and the distance of their corresponding vectors is small.

The pre-processing transfers each news in raw text into a fixed size matrix. Then we feed the processed training data into the LSTM unit to train the model. The LSTM is still a neural network. But different from the fully connected neural network, it has cycle in the neuron connections. So the previous state (or memory) of the LSTM unit c_t will play a role in new prediction h_t.

$$h_t = o_t \cdot \tanh(c_t) \tag{9}$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c_t} \tag{10}$$

$$\tilde{c_t} = \tanh(x_t W_c + h_{t-1} U_c + b_c) \tag{11}$$

$$o_t = \sigma(x_t W_o + h_{t-1} U_o + b_o) \tag{12}$$

$$i_t = \sigma(x_t W_i + h_{t-1} U_i + b_i) \tag{13}$$

$$f_t = \sigma(x_t W_f + h_{t-1} U_f + b_f) \tag{14}$$

The details of the theory of the LSTM is introduced in [8]

5 Comparison of Results

We compared our models using their Confusion Matrices to calculate the Precision, Recall and the F_1 scores. The Table 1 shows our results.

Table 1: Model Performance on the Test Set

Name	Precision	Recall	F_1
Naive Bayes	0.68	0.86	0.76
SVM	0.85	0.93	0.89
Neural Network with TensorFlow	0.77	0.92	0.84
Neural Network with Keras	0.92	0.93	0.92
LSTM	0.94	0.94	0.94

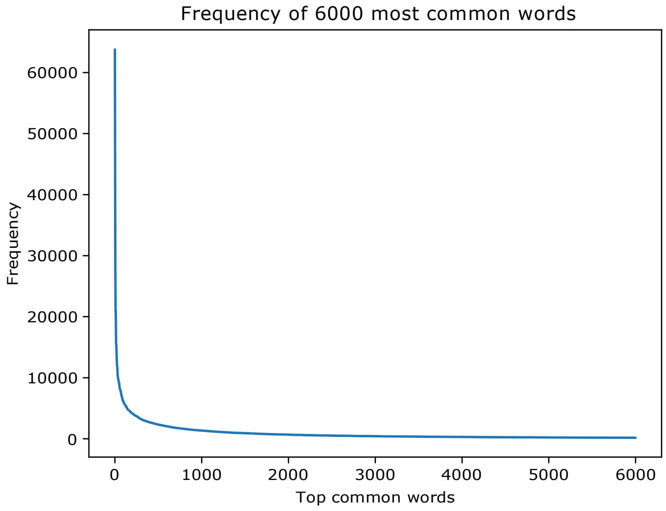

Figure 1: Frequency of Top Common Words

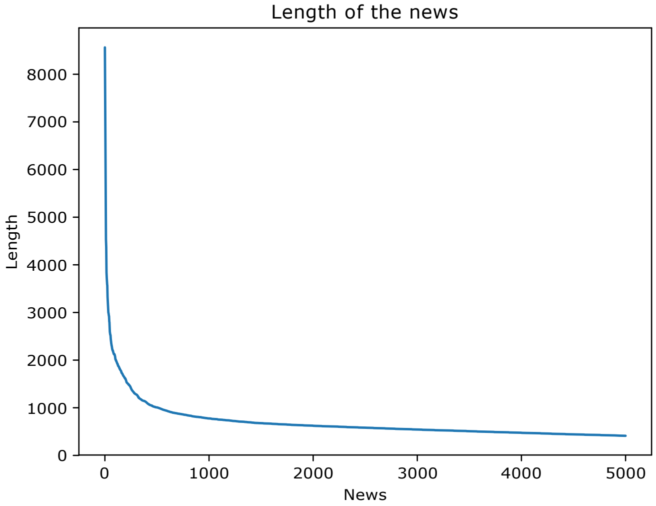

Figure 2: Length of the News

We observed that LSTM gave us the best results. We had to use a different set of embeddings for pre-processing the data to be fed to our LSTM model. It uses ordered set of Word2Vec representations.

The LSTM achieves the highest F_1 score in comparison to all the other models, followed by the Neural Network model using Keras. One of the reasons that LSTM performs so well is because the text is inherently a serialized object. All the other models use the Doc2Vec to get their feature vectors and hence, they rely on the Doc2Vec to extract the order information and perform a classification on it. On the other hand, the LSTM model preserves the order using a different pre-processing method and makes prediction based on both the words and their order. This is how it outperforms others.

6 Future Work

A complete, production-quality classifier will incorporate many different features beyond the vectors corresponding to the words in the text. For fake news detection, we can add as features the source of the news, including any associated URLs, the topic (e.g., science, politics, sports, etc.), publishing medium (blog, print, social media), country or geographic region of origin, publication year, as well as linguistic features not exploited in this exerciseuse of capitalization, fraction of words that are proper nouns (using gazetteers), and others.

Besides, we can also aggregate the well-performed classifiers to achieve better accuracy. For example, using bootstrap aggregating for the Neural Netwrok, LSTM and SVM models to get better prediction result.

An ambitious work would be to search the news on the Internet and compare the search results with the original news. Since the search result is usually reliable, this method should be more accurate, but also involves natural language understanding because the search results will not be exactly the same as the original news. So we will need to compare the meaning of two contents and decide whether they mean the same thing.

Acknowledgments

Thank You so much to Professor Sanjay Sir for good guidance to complete this report.From this topic I gained a lot of knowledge

References

[1] Zuckerberg, M., *Facebook Post*, `https://www.facebook.com/zuck/posts/10103253901916271`, November, 2016.

[2] Allcott, H., and Gentzkow, M., *Social Media and Fake News in the 2016 Election*, `https://web.stanford.edu/œgentzkow/research/fakenews.pdf`, January, 2017.

[3] Datasets, *Kaggle*, `https://www.kaggle.com/c/fake-news/data`, February, 2018.

[4] Source Code Repository, *GitHub*, `https://github.com/FakeNewsDetection/FakeBuster`, April, 2018.

[5] Quoc, L., Mikolov, T., *Distributed Representations of Sentences and Documents*, `https://arxiv.org/abs/1405.4053`, May, 2014.

[6] Christopher, M. Bishop, *Pattern Recognition and Machine Learning*, `http://users.isr.ist.utl.pt/˜wurmd/Livros/school/Bishop%20-%20Pattern%20Recognition%20And%20Machine%20Learning%20-%20Springer%20%202006.pdf`, April, 2016.

[7] Goldberg, Y., *A Primer on Neural Network Models for Natural Language Processing*, `https://arxiv.org/pdf/1510.00726.pdf`, October, 2015.

[8] Hochreiter, S., Jrgen, S., *Long short-term memory*. `http://www.bioinf.jku.at/publications/older/2604.pdf`, October, 1997.