Fake News Detection with Naïve Bayes, Random Forest, and LSTM

University of Information Technology

1st Pham Dang Khoa   
*Department of Computer Science*  
*University of Information Technology*Vietnam National University – Ho Chi Minh City, Vietnam  
Email: [18520930@gm.uit.edu.vn](mailto:18520930@gm.uit.edu.vn)

2nd Phan Hoang Phuoc  
*Department of Computer Science*  
*University of Information Technology*Vietnam National University – Ho Chi Minh City, Vietnam  
Email: [17520918@gm.uit.edu.vn](mailto:17520918@gm.uit.edu.vn)

*Abstract* - Fake news and hoaxes have overwhelmed for many decades around our lives. Since the arrival of the Internet, this problem has been more and more serious. Several social media and newspapers compete with each other by publishing or releasing junk news to gain more readers for their profits, which is a part of psychological warfare. Their main purpose is to obtain benefits from users’ click baits. As a result, there are toxic topics put into readers’ brains, people all over the world are dealing with this huge challenge.

This paper ideally detects whether the given text in a piece of particular news is fake or real. First of all, we need to perform some pre-processing steps to clean the text data and then apply feature extraction on them. Ultimately, we will train our model and compare the accuracy of techniques which are LSTM, Naïve Bayes and Random Forest to find out which is the best fit for the model.

Keywords— LSTM, word-embedding, Random Forest, Naïve Bayes, fake news (keywords).

# **INTRODUCTION**

Fake news is a status of misinformation or disinformation which is spread rapidly via mainstream social media. Especially, as I observed, fake news was overwhelming during the COVID-19 outbreak in several nations and the 2016 US election. In several situations, it was discovered that the number of shared hoax news has been more than that of accurate news. Moreover, it is understood by the fact that when a fake news is collected another will instantly share it. As a result, fake news becomes a serious problem, with a high risk of leading to misbehaving, misunderstanding, even violence to the origin of the author’s report.

It is also found that spam messages and fake news have similarities. They use manipulative ways to win over the reader’s opinions. Most of them have grammatical mistakes and they just show half of the truth to their readers. Since both the media share such similar properties, we can use similar approaches to detect fake news accurately. One way to tackle fake news is to manually classify news as real or fake. It seems like the simplest solution but it is not practical with millions of news continuously updates daily, so that is very difficult to get produced to manually label it. Hence, there is a need to look for a pragmatic technical solution to do the same. The proposed method in this research is to exploit the advancement in machine learning. To do the same, the classification model has been trained with various machine learning algorithms to label the data. The results from the study indicated neural networks to be the case in which news highest accuracy is achieved. In this paper, we will aim to take data in the testing set as input text and classify it.

# **DATA**

## **Brief description of the dataset**

Our dataset is a collection of approximately 40000 raw articles consisting of fake and real news. The number of each label seems to be the same as each other. It has been split into two separate datasets which label Fake and Real. Each data point in this dataset has four features which are title, text, subject, and date. Contents of those articles concentrate mainly on the US politics news. This dataset, obtained on Kaggle, was just raw data and so noisy that needed cleaning.

## **Preprocessing data**

These article’s contents are raw data which is a usual part of this dataset, they included many noises that we must clean. First of all, we will check the null values in our dataset, then consider which features are the most important to keep them and remove the remaining records. Moreover, we need to perform some pre-processing techniques before we can start implementing the model in our project. [1]

Here are some text pre-processing techniques we will approach and mention the role of each:

* **Lowercasing letters:** To lowercase all letters in the entire text data, which is one of the most effective but plainest methods of preprocessing text. Because in many NLP problems, we find that a huge deviation between capitalized text input and non-capitalized texts will give us significantly different outputs (ex: “USA” vs “usa”, “Many” vs “many”).
* **Stemming and Lemmatization:** These methods seem to be analogous to each other**.** Both of them are processes that try to reduce inflection in words into their original form but they still have differences. Stemming uses an extreme process that chops off the ends of those words in expectation of achieving this target correctly, sometimes derivational units are also removed. Here is an example of these techniques:

|  |  |
| --- | --- |
| **Original word** | **Stemmed or Lemmatization word** |
| troubles | trouble |
| raised | raise |
| thought | think |

* **Removing stop-words:** Stop words is a set of regularly used words in a specific language. They do not carry much meaning in the whole text and are often taken out. Examples of stop words in English are “am”, “is”, “are”, “it”. The intuition while using this method is that, we can focus only on the important words. Moreover, it helps us cut down the number of features which lets our model decently sized. We will download package stop-words with the Natural Language tool kit (NLTK) and then get rid of them.
* **Noise removal:** It is about removing some pieces of text which can intrude into our text analysis. Particularly, noise can be URLs, some superfluous digits that are not relevant to our analysis or punctuation, even the set of symbols [!”#$%&’()\*+,-./:;<=>?@[\]^\_`{|}~]. Therefore. we need to delete them as it can produce results that are inconsistent in our downstream tasks. Here is an example of that:

|  |  |
| --- | --- |
| **Raw word** | **Cleaned word** |
| @realDonaldTrump | realDonaldTrump |
| #( \_ )<) ) THAT'S / | THAT’S |
| stepsâ€ | steps |

* **Tokenization:** It is the process that splits our given text into smaller slices called *tokens.* These tokens will be stored in the form analogous to a “list of string”. We will also use the Natural Language tool kit (NLTK) library to implement this in our project.

# **METHODS**

## **Pre-trained model:**

Because data in Natural Language Processing (NLP) area is the total raw text data that machines cannot understand. Therefore, we need to transfer them into the numerical representation of words that capture their meanings, semantic relationships, and the different types of contexts they are used in. That is exactly what definition of Word Embedding is. In summary, pre-train Word Embeddings are the embeddings learned in one task that is used for solving another similar task. In this project, we use the popular one called Word2vec. The output of Word2vec is a vocabulary where each item has a vector attached to it, which can be fed into a deep learning neural network such as LSTM.

## **Prediction Models:**

In this part, we will introduce three methods for constructing our classifiers and evaluate them based on some metrics: accuracy score, recall score, precision score and F1 score. We will delve into their detailed theories below:

1. **Naïve Bayes**

This is a very simple and regular probabilistic method in machine learning classifiers which could be used for labeling. It aims to compute the conditional probability P(C|x), which is the probability of x occurring in a certain class C. C is usually assumed to be a categorical variable with two or more discrete values. The restriction of Naïve Bayes is that all features are independent mutually. The Naïve Bayes algorithm is developed based on Bayes theorem, below is the classic formula:

(1)

Based on equation (1), it can help us determine which class that data point belongs to by choosing the most highly probabilistic class.

(2)

There are many types of Naïve Bayes model under Scikit-Learn library: Gaussian Naïve Bayes, Multinomial Naïve Bayes, and Bernoulli Naïve Bayes. In this project, we planned to import MultinomialNB to fit the data in the training set as it is so popular and used for discrete counts.

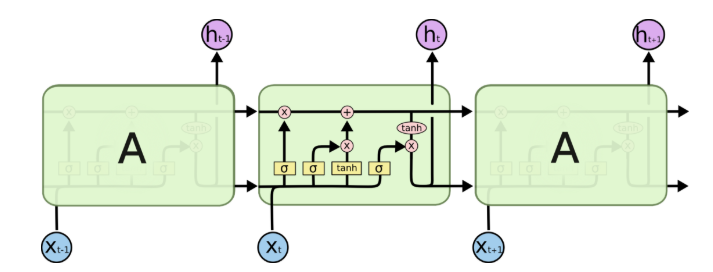
1. **LSTM**

LSTM, which stands for *Long short-term memory,* is an improved extension of the previously well-known RNN (Recurrent Neural Network) model. It includes three gates called input gate, output gate, and forget gate. The forget gate decides what information that needs to be thrown away from the cell state since these features have poor value or weight. This gate has a sigmoid function to make that decision. It looks at previous and to compute and outputs a value between 0 and 1 for each number in the cell state . An approximate 1 value represents “should keep this feature”, whereas approximate 0 value represents “should eliminate this feature”. Therefore, it has an enormous impact to be the best fit for this model. Unlike several interconnected neural network architectures, LSTM has looped neurons. Below are , and representing respectively the equation of the input, forget, and output gate. The letter *w* is symbolic of weights and the sigmoid function is represented by . We also introduce a figure that demonstrates how LSTM works.

(3)

(4)

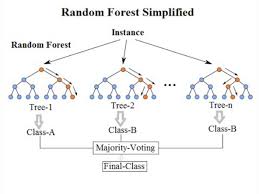
(5)



**Figure 1: The repeating module in an LSTM contains four interacting layers. [4]**

1. **Random Forest**

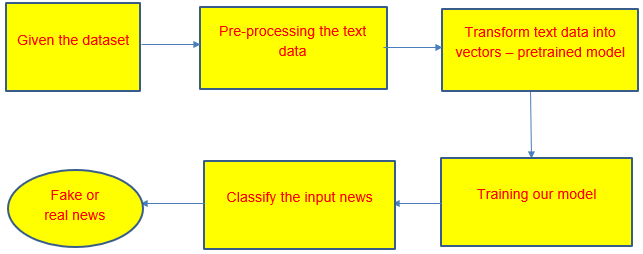
Random forest is a supervised learning algorithm used for both classification and regression, which is an ensemble of hundreds or thousands of decision trees. The core idea of the Random forest is the wisdom of crowds. Each tree plays a role of class predictor and the predictor has the most votes will decide our final model’s prediction. This algorithm aims to handle bias-variance tradeoff balanced. Consequently, there are ways to have a diverse group of classifiers, which are to split the training set into many random subsets and train data on them **[8]**, called bagging and pasting. As a result, this will lead to a low correlation of trees. Concurrently, each individual tree in the *forest* splits its branch into new branches until leaf nodes in order to optimize the MSE, then the algorithms have achieved both low bias and even low variance. The crucial characteristic is that the predictions of each tree must have a low correlation with each other because they can produce ensemble predictions that are more accurate than any of the individual predictions. The basic explanation for this effect is that the right trees will protect and fix some other wrong trees, as long as they are not continuously wrong in the same direction. Therefore, a group of trees can move in the correct direction.



**Figure 2: Demonstration of Random Forest.**

# **CLASSIFICATION PROCESS**

In this section, we will summarize how our classifier works by a short representation of a set of blocks. In the first step, we performed pre-processing data by techniques mentioned above which includes stopwords and noise removal, lowercasing the whole text as well as stemming and lemmatizing. After that, the word embedding method called Word2Vec was implemented to transform our *clean* text into vectors which represent the documents involved and they will be put into our model to train, especially useful for LSTM technique. Ultimately, the classifier we built will predict and decide the given news belongs to the real or fake class. Below is the presentation we display to conclude what we have done during this classification process:



**Figure 3: Our classification process**

# **RESULTS**

Table 1 below displays the results by calculating the accuracies as well as evaluation metrics of various models we have mentioned before. Those values shown in the table are the values when we implemented models in the test set.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Accuracy** | **Precision** | **Recall** | **F1-score** |
| LSTM | 98.74% | 0.99 | 0.98 | 0.985 |
| Naïve Bayes | 93.1% | 0.93 | 0.93 | 0.93 |
| Random Forests | 99.47% | 0.99 | 0.99 | 0.99 |

**Table 1: Evaluation of models in the testing set**

# **ERROR ANALYSIS**

Despite good values in the table, we have found some potential disadvantages in our classification task. Suppose that we feed the text which is labeled real and of course, it will output “Real news”. However, we tried modifying a character’s name in this text, it still keeps its output as the previous one. We thought that because people’s names have low weights, our model tends to ignore it and leaves an error.

# **CONCLUSION AND FUTURE DEVELOPMENT**

In conclusion, we have conducted and presented a model to detect fake news via three different Machine Learning methods. Additionally, our paper analyzes these techniques and compares them with popular metrics. The model which obtains the highest accuracy is Random Forest with the accuracy score is 99.47% in the test set.

Fake news detection has been a developing research field that has several available datasets. However, some datasets have a high bias and their labels are not sometimes unreliable as they depend on the owner’s mindset or viewpoints.

In our future development, we will try to collect more data from other areas to enrich the content for our model to train more effectively as well as get experts manually to label them firstly.

##### **References**

1. [*https://www.kdnuggets.com/2018/03/text-data-preprocessing-walkthrough-python.html*](https://www.kdnuggets.com/2018/03/text-data-preprocessing-walkthrough-python.html)
2. Poonam Tijare, Prannay S Reddy, Diana Elizabeth Roy, *P. Manoj, M. Keerthana, “A Study on Fake News Detection Using Naïve Bayes, SVM, Neural Networks and LSTM”.*
3. Pengfei Liu, Xipeng Qiu, Xuanjing Huang, “Recurrent Neural Network for Text Classification with Multi-Task Learning”.
4. Kelly Stahl, California State University Stanislaus, “Fake news detection in social media”.
5. <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>
6. Hands-on Machine Learning with Scikit-Learn Book.
7. <https://towardsdatascience.com/understanding-random-forest-58381e0602d2>
8. Ho, Tin Kam (1998). "The Random Subspace Method for Constructing Decision Forests" (PDF). IEEE Transactions on Pattern Analysis and Machine Intelligence. 20 (8): 832–844. doi:10.1109/34.709601.
9. An Essential Guide to Pretrained Word Embedding