Fake News Project Report

Project Title - Fake News Detection

Group Members: Sidorela Mema, Zuhal Tas-Batirbek

Dataset:
https://www.kaggle.com/datasets/saurabhshahane/fake-news-classification

Our google colab notebook:
https://colab.research.google.com/drive/1bWGrvbh7O3rUWKC5NBD1OqD6cEOM1QwK?usp=sharing

Fake News Classification Dataset:

Usability Score: 10,
Published in: IEEE Transactions on Computational Social Systems
Size: 72K news articles
Features: Title, Text, Label (0-1)

The Welfake dataset is a collection of news articles labeled by professional journalists as either true or false. The dataset contains over 72,000 articles from different news sources and topics. The articles were collected between October 2016 and November 2017.

Introduction

The spread of fake news has become a significant concern in today's digital age, as it can have a detrimental impact on public opinion and trust. To address this issue, we present a machine learning project to classify news articles as either real or fake using the Welfake dataset.

Loading Dataset

Uploading the dataset into Google Collab was a challenge, since our dataset is hosted on an external platform such as Kaggle. The issue that arises when uploading a dataset from Kaggle is authentication errors, where the user is prompted to enter their Kaggle API credentials in order to download the dataset. It did not seem like a good idea. We tried to upload them to Github and read it from there but the files were too big to upload to Github.
To avoid authentication and other issues we uploaded the dataset to Google Drive and access it from Google Collab using the Google Drive API. With the lines below, we easily did the Google Authentication step.

```
from google.colab import drive
drive.mount('/content/drive')
```

Data Preprocessing

We preprocess the text data by doing some data cleaning by removing URLs, non-ASCII characters, numbers, punctuations, and stop words.

Next we did tokenizing, and lemmatizing the words to obtain a normalized representation of the articles. We do tokenization first to break down the text into smaller units and identify relationships between them, and then lemmatization to reduce them to their base forms and better understand their meaning.

Vectorizing Data

We have to transform the words to numbers, because computers can't read words. Texts that turned into space-separated sequences of words are split into lists of tokens. They will then be indexed or vectorized.

1. Bag of Words: Countvectorizer

The most basic and naive method of transforming words into vectors by counting occurrences of each word in each document. Bag of Words lists words paired with their word counts per document. Before they're fed to the neural network, each vector of word counts is normalized such that all elements of the vector add up to one. That's why, the frequency of each word is converted to represent the probabilities of those words' occurrence in the document. Probabilities that surpass certain levels will activate nodes in the network and influence the document's classification.

2. TF-IDF :

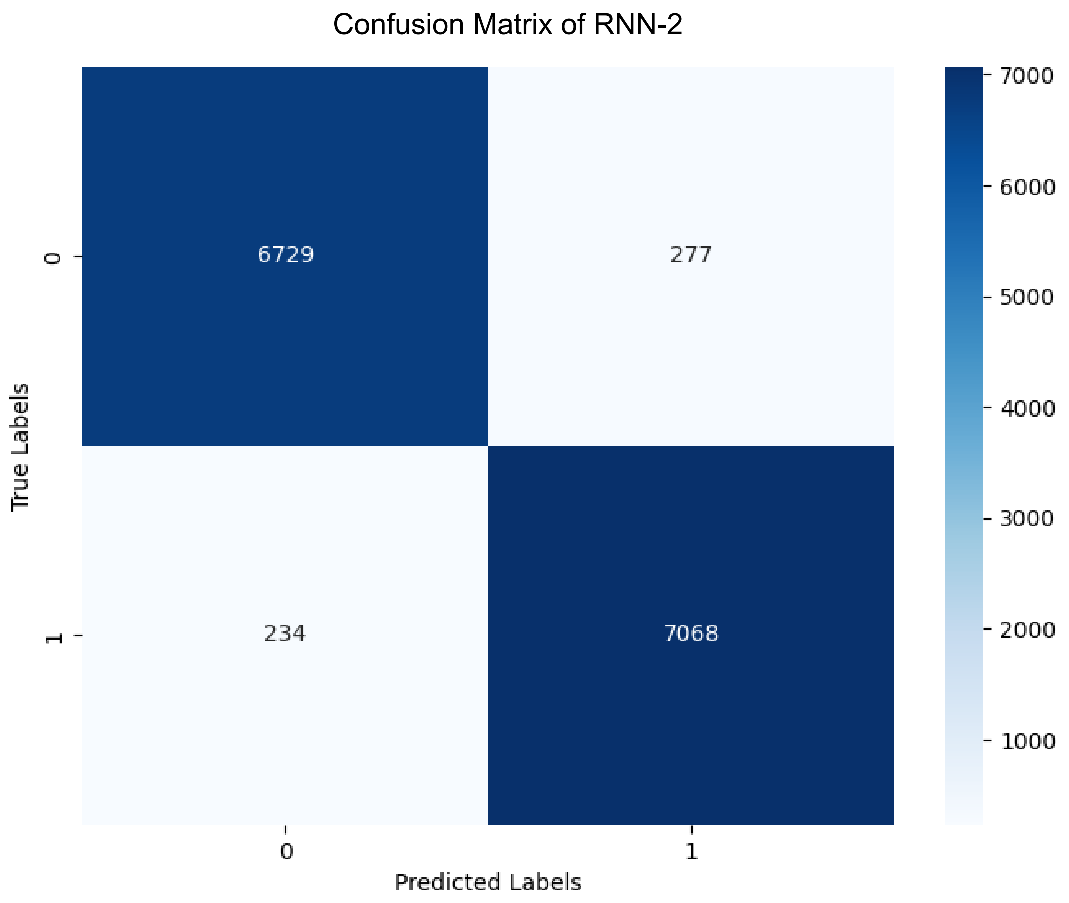
Term Frequency and Inverse Document Frequency is used to find the related content and important words and phrases in a larger text.
The more documents a word appears in, the less valuable that word is as a signal to differentiate any given document. The aim is to leave only the frequent AND distinctive words as markers. TF-IDF is also normalized and adds up to one.

TF: It's a measure of a word w in a document d. It is equal to the number of instances of word w in document d divided by the total number of words in document d.
IDF: Inverse Document Frequency of word w is defined as the total number of documents (N) in a text corpus D, divided by the number of documents containing w. It gives a numeric value of the importance of a word.

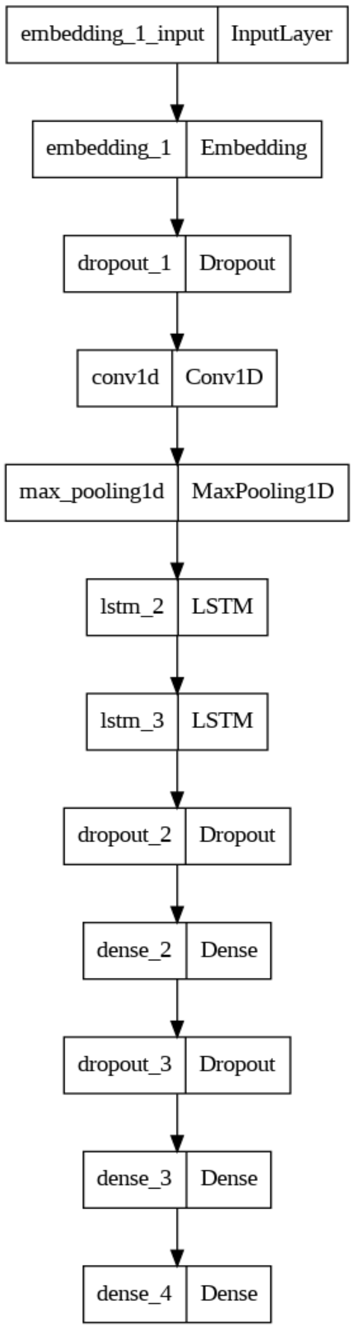
The product of TF and IDF is the TF-IDF. TF-IDF is usually one of the best metrics to determine if a term is significant to a text. It represents the importance of a word in a particular document.

Models

Model	Training Accuracy	Test Accuracy
Logistic Regression	1.0	0.95
Naive Bayes	0.90	0.89
Random Forest	0.95	0.94
RNN -1	0.98	0.96
RNN - 2	0.99	0.96
CNN + RNN	0.99	0.97



Combination of CNN and RNN



After the words are converted into word embeddings, the words are fed into a neural network. This neural network consists of various layers;

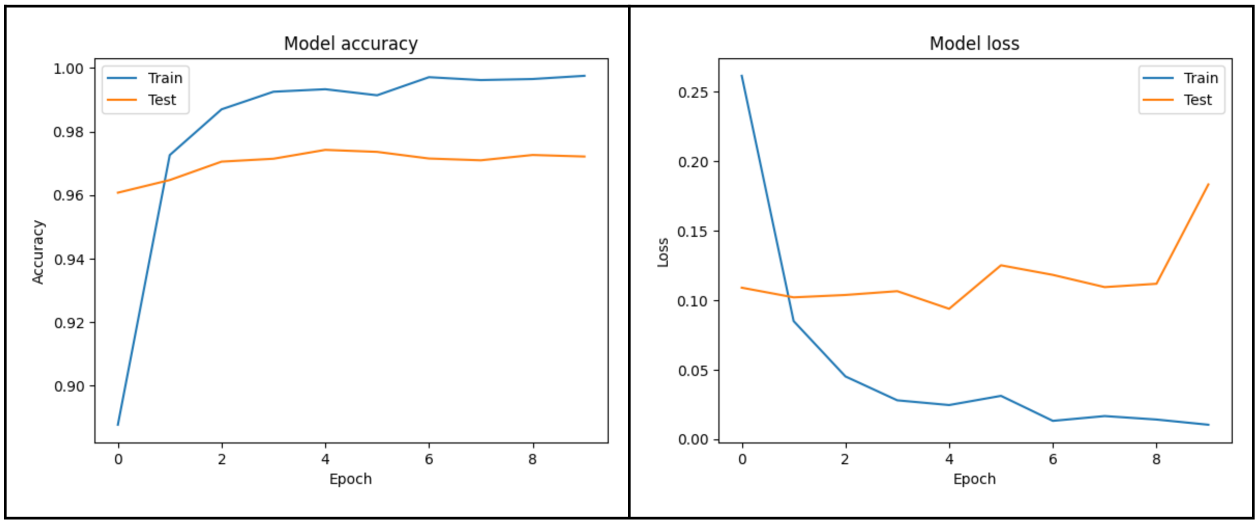
The first layer is a convolutional layer. In image detection, convolutions are used to detect edges or shapes. In Natural Language Processing convolutions can improve performance.

The next layer is a max pooling layer. This layer iterates over the tensors and takes the highest value. In this way the feature space is compressed. This step makes sure important features are kept, while empty space is dropped.

The next layer is a Long Short Term Memory (LSTM) Layer.

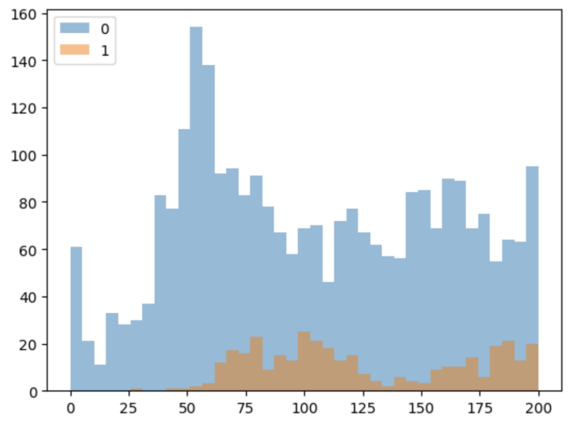
The last layer before the prediction is made is a fully connected layer. This is a regular neural network layer where all nodes are connected with each other.

Training and test accuracy and loss over each epoch

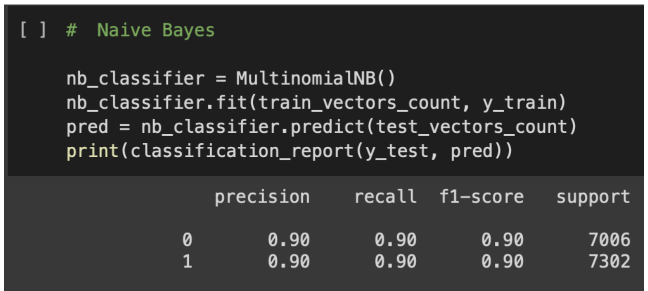


Practical Results Obtained

1. Our analysis revealed a clear and statistically significant difference between the length of news articles for real and fake news. We visualized this difference using a histogram, which clearly showed a divergence in the distribution of article lengths between the two categories. This finding is significant, as it suggests that the length of a news article may be a useful feature for distinguishing real and fake news.



2. While people on Kaggle scored between 80% to 85% for these simple models, we scored between 90% to 95%.

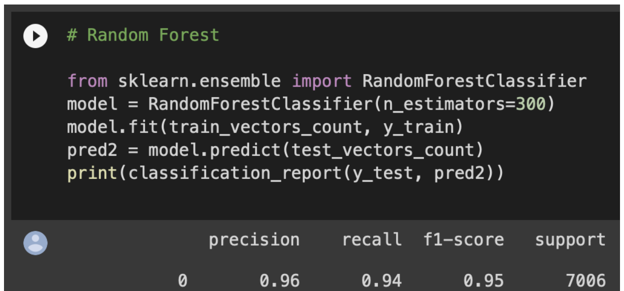


```
[ ]  #  Naive Bayes

     nb_classifier = MultinomialNB()
     nb_classifier.fit(train_vectors_count, y_train)
     pred = nb_classifier.predict(test_vectors_count)
     print(classification_report(y_test, pred))

                   precision    recall  f1-score   support

              0       0.90      0.90      0.90      7006
              1       0.90      0.90      0.90      7302
```



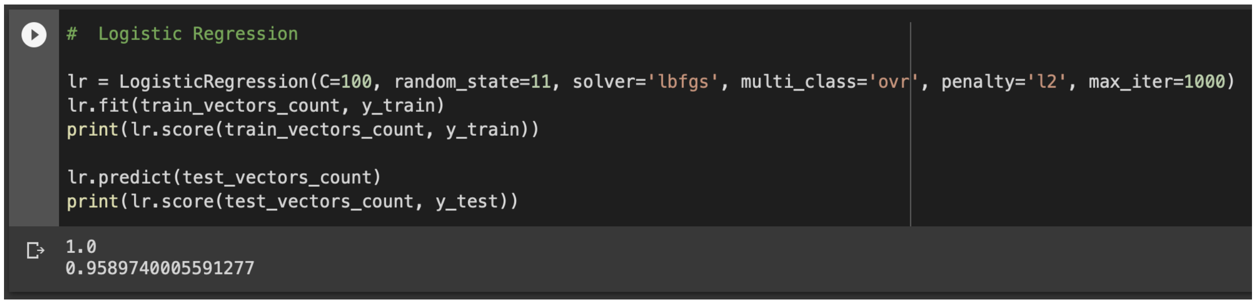
```
# Random Forest

from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=300)
model.fit(train_vectors_count, y_train)
pred2 = model.predict(test_vectors_count)
print(classification_report(y_test, pred2))

              precision    recall  f1-score   support

           0       0.96      0.94      0.95      7006
```

3. We got 100% score for the training dataset, however it is not common for logistic regression to achieve 100% accuracy on a training set, especially on a large and diverse dataset like WELFake.



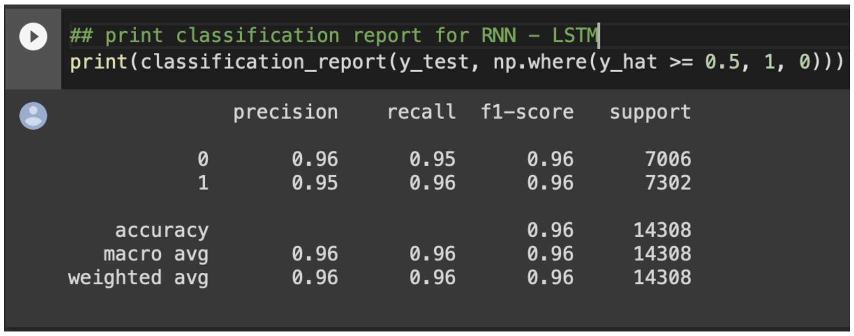
```
#  Logistic Regression

lr = LogisticRegression(C=100, random_state=11, solver='lbfgs', multi_class='ovr', penalty='l2', max_iter=1000)
lr.fit(train_vectors_count, y_train)
print(lr.score(train_vectors_count, y_train))

lr.predict(test_vectors_count)
print(lr.score(test_vectors_count, y_test))
```

```
1.0
0.9589740005591277
```

4. RNN score was very close to Random Forest score which was 95%



```
## print classification report for RNN - LSTM
print(classification_report(y_test, np.where(y_hat >= 0.5, 1, 0)))
```

```
              precision    recall  f1-score   support

           0       0.96      0.95      0.96      7006
           1       0.95      0.96      0.96      7302

    accuracy                           0.96     14308
   macro avg       0.96      0.96      0.96     14308
weighted avg       0.96      0.96      0.96     14308
```

What Differentiates our Work from Previous Works on Kaggle

What was interesting about our work is seeing that our RNN model converged faster than others'. We experimented with two different RNN models, each with different hyperparameters. We found out that one model converged in just four epochs, while the other one took only eight epochs while previous works on Kaggle stopped in 10 epochs which was the maximum iteration number. We think that data preprocessing and cleaning may make up the difference. Not all notebooks did data processing before applying the models. Some notebooks only include the data processing part while others only include the modeling part.

Conclusion

Overall modeling fake news dataset is a complex and challenging task and machine learning can be an effective tool to quickly analyze large amounts of data and identify patterns that are difficult for humans. We experimented with popular algorithms such as random forest, Naive Bayes, logistic regression, and deep learning techniques. We achieved the highest accuracy of 97% by using a combination of RNN and CNN algorithms. Nonetheless, the performance of the random forest algorithm was also impressive, with an accuracy rate of 94% that came close to RNN and CNN's accuracy.

References

https://towardsdatascience.com/how-to-build-a-recurrent-neural-network-to-detect-fake-news-35953c19cf0b

https://github.com/matdekoning/FakeNewsClassifier/blob/master/FakeNewsClassifier.ipynb

http://wiki.pathmind.com/bagofwords-tf-idf

https://www.analyticsvidhya.com/blog/2021/09/creating-a-movie-reviews-classifier-using-tf-idf-in-python/