

# Fake News Detection via Bert

**By**

**Asem Okby**

## Table of Contents

|                               |           |
|-------------------------------|-----------|
| <b>Abstract</b>               | <b>3</b>  |
| <b>Data Discovery</b>         | <b>3</b>  |
| <b>Cleaning The Data</b>      | <b>5</b>  |
| <b>Vectorization</b>          | <b>6</b>  |
| TFIDF                         | 7         |
| Word2Vec                      | 7         |
| Bert                          | 7         |
| <b>Preprocessing The Data</b> | <b>8</b>  |
| Scaling                       | 8         |
| Dimensionality Reduction      | 8         |
| <b>Modelling</b>              | <b>8</b>  |
| Setup                         | 8         |
| Results                       | 9         |
| All Data                      | 9         |
| PCA                           | 9         |
| Ensemble Method               | 10        |
| Hard Voting                   | 10        |
| Soft Voting                   | 11        |
| Hard Vs. Soft Voting          | 11        |
| <b>Conclusion</b>             | <b>11</b> |
| <b>References</b>             | <b>12</b> |

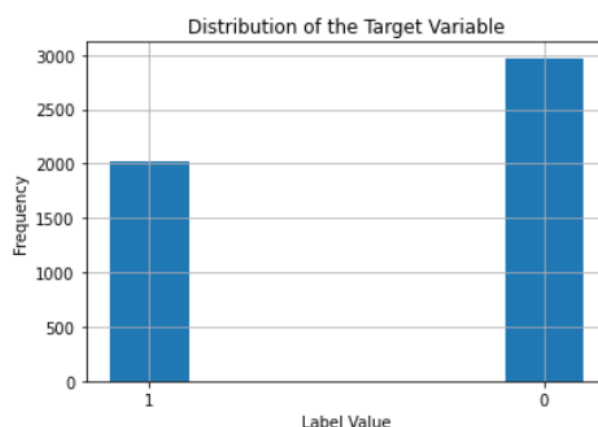
# Abstract

Fake news is false or misleading information. False information can cause several problems because people and even governments could take actions based on fake news. And some actions could lead to very serious losses. Therefore, detecting whether published news is fake or not is a serious task, especially nowadays where we live in a small world where everyone can stream news to the world in a matter of seconds. In this project, we address the fake news detection problem, classifying a given news as normal or fake.

Our dataset is taken from a Kaggle challenge [1]. It is composed of 2 features. A feature containing the news, a text, and a binary feature containing either 0 or 1 indicating whether the text is fake news or not. We process the text using different embedding techniques. Basic ones such as TF-IDF, and more sophisticated ones such as Word2Vec and Bert. Once we get the embeddings, we build various machine learning classification models such as Logistic Regression, Support Vector Machines, and RandomForestClassifier. Besides, we ensemble more than a classifier together using a voting system. Finally, we interpret our results and discuss possible improvements.

## Data Discovery

In this section, we understand, analyze and visualize the data to gain insights that could help us with some decisions taken later while processing and modeling the data. We start off with the target variable. Figure 1, which shows the distribution of the target variable, shows that the data is not balanced. We keep this in mind when evaluating the performance of the models and avoid some metrics that would mislead us.



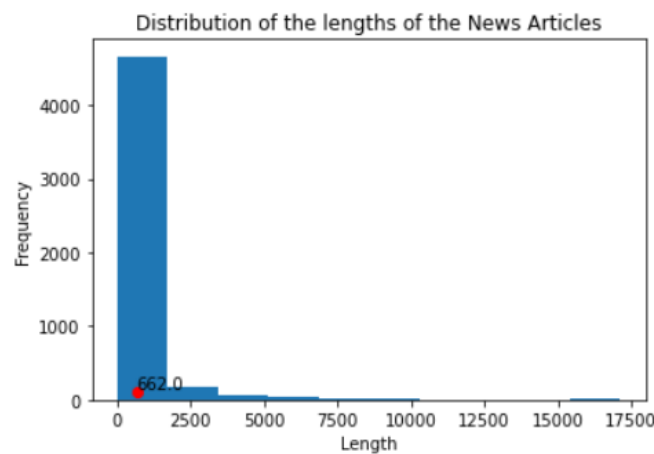
**Figure 1:** Distribution of the target variable.

We also look at the value counts of the variable, and we notice that there is a single value called label. Before running into conclusions, we look at the row itself, Figure 2, and notice that it has no feature value, which indicates that the row is junk, therefore, we drop it.

| text         | label |
|--------------|-------|
| 1615 content | label |

**Figure 2:** A row with a label value "label".

Next, we look at the single feature we have, text, which contains news articles. We start with learning some statistics about it e.g. the average length of a news article. Figure 3 shows that most of the lengths are below 2500. This requires us later to have a fixed size vector representation later by truncating the news articles to a specific length. The red annotated point in Figure 3 shows the average length.



**Figure 3:** Distribution of the lengths of the news articles.

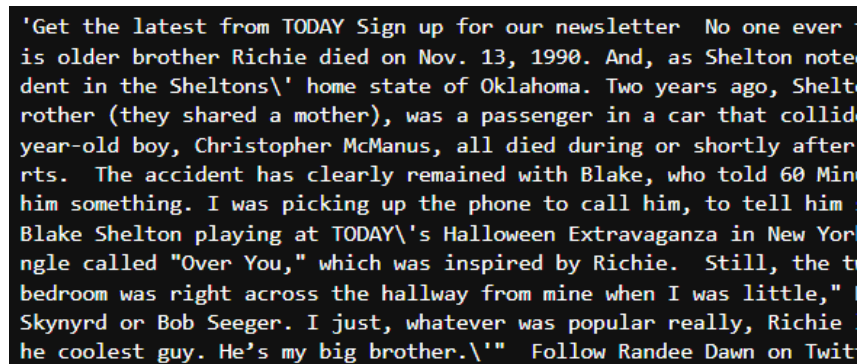
We also look at the highest probable Bigrams, sequences of two adjacent words, in the normal articles and the fake articles. Figure 4 shows the top 5 Bigrams and their probabilities in the normal and fake news articles. Most of them are stop-words, which are very common in the English language and don't carry useful information. Similarly, most of the top 5 Bigrams in the fake articles are stop-words. We expect to see more useful Bigrams after cleaning the data, which we do in the following section.

| bigram | probability | bigram | probability |
|--------|-------------|--------|-------------|
| at the | 0.183098    | at the | 0.189514    |
| to the | 0.202521    | to the | 0.193482    |
| on the | 0.211595    | on the | 0.217805    |
| in the | 0.407152    | in the | 0.430540    |
| of the | 0.473564    | of the | 0.438623    |

**Figure 4:** Left: the top 5 highest probable bigrams in the normal articles.  
Right: the top 5 highest probable bigrams in the fake articles.

# Cleaning The Data

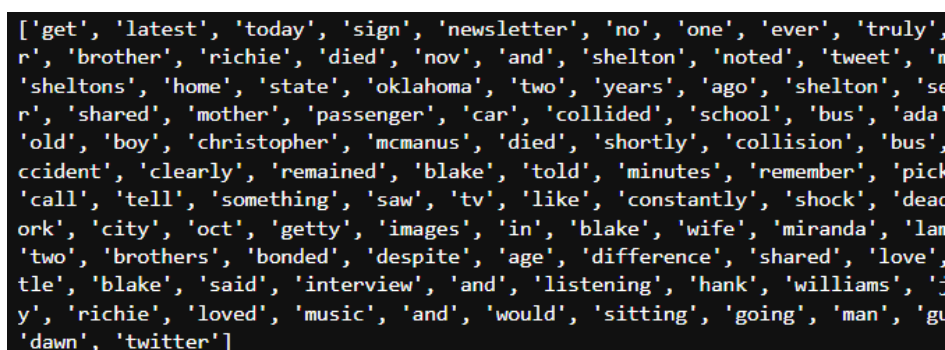
The text feature, the articles, are represented as raw text. The text needs to be cleaned before vectorizing it to get rid of the tokens/words that would hurt the model. Tokens like punctuations, quotation marks, links..etc. Such tokens could actually contribute to the target's value e.g. fake news may overuse punctuations, however, we assume they don't carry any information and eliminate them. Later, we may engineer some features that would represent such tokens, e.g. a statistic about punctuations, to help us improve the performance.



```
'Get the latest from TODAY Sign up for our newsletter No one ever  
is older brother Richie died on Nov. 13, 1990. And, as Shelton note  
dent in the Sheltons\' home state of Oklahoma. Two years ago, Shelt  
rother (they shared a mother), was a passenger in a car that collid  
year-old boy, Christopher McManus, all died during or shortly after  
rts. The accident has clearly remained with Blake, who told 60 Min  
him something. I was picking up the phone to call him, to tell him  
Blake Shelton playing at TODAY\'s Halloween Extravaganza in New Yor  
ngle called "Over You," which was inspired by Richie. Still, the t  
bedroom was right across the hallway from mine when I was little,"  
Skynyrd or Bob Seeger. I just, whatever was popular really, Richie  
he coolest guy. He's my big brother.\" Follow Randee Dawn on Twit
```

Figure 5: A snippet of a part of an article.

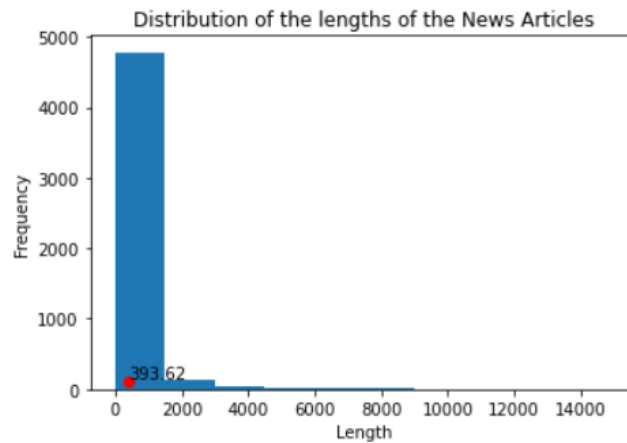
The text contains some junk that we need to clean. We clean the text in the following steps: 1) Removing punctuations. 2) Removing digits. 3) Removal of the empty strings, the one character tokens, and stop-words. Figure 6 shows a snippet of a part of an article after being cleaned.



```
['get', 'latest', 'today', 'sign', 'newsletter', 'no', 'one', 'ever', 'truly',  
'r', 'brother', 'richie', 'died', 'nov', 'and', 'shelton', 'noted', 'tweet', 'n  
'sheltons', 'home', 'state', 'oklahoma', 'two', 'years', 'ago', 'shelton', 'se  
'r', 'shared', 'mother', 'passenger', 'car', 'collided', 'school', 'bus', 'ada  
'old', 'boy', 'christopher', 'mcmanus', 'died', 'shortly', 'collision', 'bus',  
ccident', 'clearly', 'remained', 'blake', 'told', 'minutes', 'remember', 'pick  
'call', 'tell', 'something', 'saw', 'tv', 'like', 'constantly', 'shock', 'dead  
ork', 'city', 'oct', 'getty', 'images', 'in', 'blake', 'wife', 'miranda', 'lan  
'two', 'brothers', 'bonded', 'despite', 'age', 'difference', 'shared', 'love',  
tle', 'blake', 'said', 'interview', 'and', 'listening', 'hank', 'williams', 'j  
'y', 'richie', 'loved', 'music', 'and', 'would', 'sitting', 'going', 'man', 'gu  
'dawn', 'twitter']
```

Figure 6: A snippet of a part of an article after being cleaned.

After cleaning the data, we notice the change in the average length of an article and the distribution of the lengths' values, see Figure 7. However, we still see some big values on the right.



**Figure 7:** Distribution of the lengths of the news articles after cleaning the data.

Finally, we look one more time at the highest bigrams in both the normal and the fake articles separately. Figure 8 shows that the bigrams now are more valuable, free of stopwords. However, they are very similar, that is why they won't help us in this problem. However, looking at the bigrams can be really helpful. They give insights about the data. Besides, they can be used as additional features.

| bigram        | probability | bigram       | probability |
|---------------|-------------|--------------|-------------|
| los angeles   | 0.141011    | brad pitt    | 0.143836    |
| united states | 0.153112    | getty images | 0.151739    |
| getty images  | 0.233866    | los angeles  | 0.184299    |
| year old      | 0.270415    | year old     | 0.236460    |
| new york      | 0.332154    | new york     | 0.270601    |

**Figure 8:** Left: the top 5 highest probable bigrams in the normal articles.  
Right: the top 5 highest probable bigrams in the fake articles.

## Vectorization

At this point, our data is clean. Now we need to represent each news article as an embedding vector. Vectorization can be done in many ways. There are simple methods such as TF-IDF and sophisticated methods such as Word2Vec and Bert. We assume that Bert embeddings would perform better than the simple TF-IDF embeddings. We will use the model trained with TF-IDF embeddings as the baseline and benchmark the models trained with Bert embeddings against it. We also use Word2Vec embeddings besides the former two methods.

## TFIDF

TF-IDF stands for Term Frequency and Inverse Document Frequency. Term Frequency is basically the frequency of a word in a document. We could use the frequency alone as a weight for each word. However, this is misleading because the frequency of a word does not necessarily tell us about the weight of the term. TFIDF handles this by normalizing these frequencies by the inverse document frequency, which captures the rarity of the word. Due to the simplicity of the method, it does not capture any semantics of the words. Besides, it produces a sparse matrix of size  $N \times V$ , where  $n$  is the number of documents and  $V$  is the size of the vocabulary, all the unique words in the text.

## Word2Vec

Word2Vec is a more sophisticated approach than TFIDF. It tries to learn an embedding vector for a word from its context. A model tries to predict a word given its context. In the end, we don't use those probabilities, but the weights of the neural networks as vector representations of the words. Word2Vec produces a dense vector. The size is generally 100-300. The embedding vectors carry semantic information. Hence, applying vector operations on the embeddings vectors produces meaningful results.

We build a Word2Vec model using our data. The trained model can be used for embedding a single word. For our case, we need a vector representation for the whole document, not just for a single word in the document. To tackle this problem, we get the embedding of each word then average all the vectors to end up with a single vector representing the document.

## Bert

Bert is a context-dependent model. It captures all the senses of a word. The input to the model is not a single word but a sentence. The vector representation of a word depends on the sense of the word in that context. Word2Vec, on the other hand, would represent a word with a single vector for all the senses of the word. Besides, Bert takes into account the position of the word unlike Word2Vec and TFIDF.

As we mentioned earlier, it takes as an input a sentence. That sentence's length should not exceed 512. And as we have seen above, some documents' length is way more than this. Therefore, we split such long documents into chunks to be able to input them to Bert. Then, as we did with Word2Vec, we get the embedding vector for each word in a document and average them to have a single embedding representation for the document.

# Preprocessing The Data

Before using the embeddings to train our models, we organize, preprocess, and split the data.

## Scaling

Scaling the data before feeding it to Machine Learning models is usually encouraged and it is sometimes even a must for some algorithms. We scale/standardize our data by removing the mean and scaling to unit variance.

## Dimensionality Reduction

The high dimensional data affects the algorithms in many ways, especially if the amount of data is not big. This problem, curse of dimensionality, is related as well to the overfitting of the models. Here we use the Principal Component Analysis (PCA) algorithm to reduce the dimension of our data, especially for the data gained using TF-IDF which is of dimension  $V$ , where  $V$  is the vocabulary size.

## Modelling

### Setup

Here, we train our models on the embeddings we have. We train several classification methods to compare their results. We also compare the results of the models trained on the actual data and the data transformed with PCA. Besides, we combine the best performing models of different embedding techniques (TFIDF, Word2Vec, and Bert) and have a voting system for their output. Finally, we interpret the results and discuss some possible improvements.

As an evaluation metric. We use two metrics: Accuracy Score and F1 Score. We mentioned earlier that our data is imbalanced, therefore, a metric like f1 score is suitable for such cases. We use the Accuracy Score as well because Kaggle uses it to evaluate the submitted results for this challenge.



## Results

### All Data

Figure 9 shows the performance of the three models trained on every embedding. The data used for the models in the first tables is the TF-IDF embeddings. We provide the validation, training and test scores of each model. We prefer to compare the models based on the F1 validation score since as we mentioned earlier the data is imbalanced. Logistic Regression's F1 validation score is the highest score among the models trained on the TF-IDF embeddings. Notice how bad the other two models are performing, their F1 validation scores.

|                        | Accuracy Score (CV=3) | Accuracy Score (Train) | Accuracy Score (Test) | F1_Score (CV=3) | F1_Score (Train) | F1_Score (Test) |
|------------------------|-----------------------|------------------------|-----------------------|-----------------|------------------|-----------------|
| Logistic Regression    | 0.740471              | 0.802909               | 0.764529              | 0.571098        | 0.697460         | 0.643399        |
| LinearSVC              | 0.657724              | 0.696841               | 0.678357              | 0.266510        | 0.408223         | 0.397749        |
| RandomForestClassifier | 0.633902              | 0.639920               | 0.608216              | 0.155641        | 0.183163         | 0.129176        |
|                        | Accuracy Score (CV=3) | Accuracy Score (Train) | Accuracy Score (Test) | F1_Score (CV=3) | F1_Score (Train) | F1_Score (Test) |
| Logistic Regression    | 0.738216              | 0.785607               | 0.765531              | 0.655043        | 0.712024         | 0.692105        |
| LinearSVC              | 0.746742              | 0.786359               | 0.766533              | 0.660933        | 0.708618         | 0.691391        |
| RandomForestClassifier | 0.742478              | 0.792377               | 0.741483              | 0.599013        | 0.675803         | 0.603077        |
|                        | Accuracy Score (CV=3) | Accuracy Score (Train) | Accuracy Score (Test) | F1_Score (CV=3) | F1_Score (Train) | F1_Score (Test) |
| Logistic Regression    | 0.700098              | 0.844032               | 0.735471              | 0.628242        | 0.800000         | 0.674074        |
| LinearSVC              | 0.718655              | 0.834504               | 0.751503              | 0.643527        | 0.784173         | 0.685279        |
| RandomForestClassifier | 0.699348              | 0.776830               | 0.689379              | 0.477175        | 0.633443         | 0.461806        |

Figure 9: Results of the three models on each of the embeddings. TF-IDF, Word2Vec, then Bert.

The models in the middle table are trained on the Word2Vec embeddings. LinearSVC's F1 validation score is higher than that of the other models. Actually, it is the highest F1 validation score in all tables.

The models in the third table are trained on the Bert embeddings. LinearSVC again is performing better than the other models. We notice that the models are overfitting as well. One of the reasons is that the size of the embedding vector of Bert is 768.

### PCA

The embedding vectors' size is at least 300 in the case of Word2Vec, and it is the size of the Vocabulary in the case of TF-IDF. This high dimensional data may hurt the performance. Here, we use the Principal Component Analysis algorithm to reduce the dimensionality, while keeping .95 variance, of the data then retrain the models again. Figure 10 shows the performance of the three models trained on every embedding.

|                        | Accuracy Score (CV=3) | Accuracy Score (Train) | Accuracy Score (Test) | F1_Score (CV=3) | F1_Score (Train) | F1_Score (Test) |
|------------------------|-----------------------|------------------------|-----------------------|-----------------|------------------|-----------------|
| Logistic Regression    | 0.742477              | 0.800401               | 0.765531              | 0.575510        | 0.693846         | 0.645455        |
| LinearSVC              | 0.657472              | 0.695587               | 0.678357              | 0.267165        | 0.405485         | 0.397749        |
| RandomForestClassifier | 0.677532              | 0.746239               | 0.691383              | 0.382667        | 0.542909         | 0.465278        |
|                        | Accuracy Score (CV=3) | Accuracy Score (Train) | Accuracy Score (Test) | F1_Score (CV=3) | F1_Score (Train) | F1_Score (Test) |
| Logistic Regression    | 0.740472              | 0.768054               | 0.761523              | 0.647678        | 0.682021         | 0.680108        |
| LinearSVC              | 0.742979              | 0.765045               | 0.761523              | 0.647398        | 0.673633         | 0.673973        |
| RandomForestClassifier | 0.721414              | 0.757773               | 0.719439              | 0.506109        | 0.579634         | 0.522184        |
|                        | Accuracy Score (CV=3) | Accuracy Score (Train) | Accuracy Score (Test) | F1_Score (CV=3) | F1_Score (Train) | F1_Score (Test) |
| Logistic Regression    | 0.721165              | 0.785105               | 0.741483              | 0.636818        | 0.717255         | 0.673418        |
| LinearSVC              | 0.724174              | 0.783099               | 0.741483              | 0.639045        | 0.712720         | 0.665803        |
| RandomForestClassifier | 0.638415              | 0.696088               | 0.638277              | 0.186821        | 0.389113         | 0.246347        |

**Figure 10:** Results of the three models on each of the transformed(PCA) embeddings. **TF-IDF, Word2Vec, then Bert.**

We can notice the effect of reducing the dimensionality for some models. For example, the F1 validation scores for some models increased. Perhaps the most obvious one is the F1 validation score of the RandomForestClassifier which is trained on the TF-IDF. The model is able to learn better in the low dimensional data. Another thing we notice is the decrease of the training time.

## Ensemble Method

Here we combine 3 models, each is the best performing model on one of the 3 embeddings we use, TF-IDF, Word2Vec, and Bert. We use a voting system to decide the final output of the three models.

### Hard Voting

Here, we decide the final output of the three models based on the most frequent output of the 3. Figure 11 shows the performance using the hard voting system.

|          | accuracy score (train) | accuracy score (test) | f1 score (train) | f1 score (test) |
|----------|------------------------|-----------------------|------------------|-----------------|
| ensemble | 0.777834               | 0.755511              | 0.678753         | 0.651429        |

**Figure 11:** The performance of the ensemble method using the hard voting system.

## Soft Voting

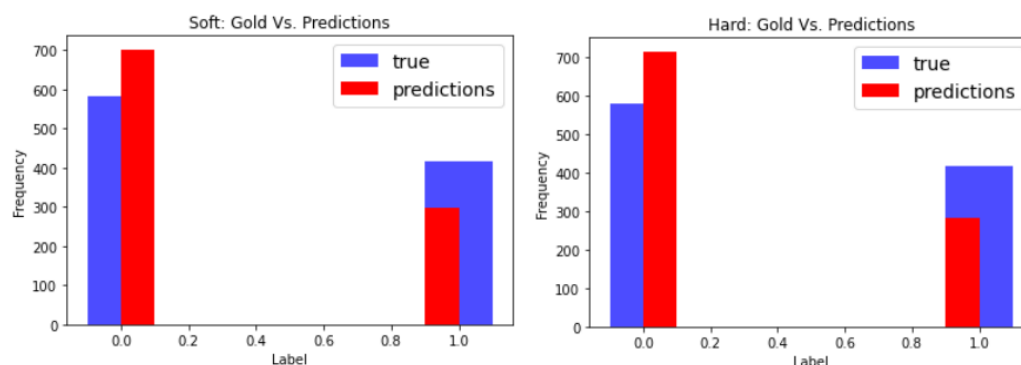
Here, we decide the final output of the three models based on the average probability of all the output probability of the three models. Figure 12 shows the performance of the ensemble method using the soft voting system.

|          | accuracy score (train) | accuracy score (test) | f1 score (train) | f1 score (test) |
|----------|------------------------|-----------------------|------------------|-----------------|
| ensemble | 0.787362               | 0.762525              | 0.692976         | 0.667602        |

**Figure 12:** The performance of the ensemble method using the soft voting system.

## Hard Vs. Soft Voting

Soft Voting is outperforming hard voting. Figure 13 shows the distribution of the target variable both using the soft and hard voting systems. The red bars on the left, which represent the predictions using the soft system, seem to mimic the blue bars better than the red bars on the right side. To be more specific, the hard voting's method misclassified 244 instances while the soft voting's method misclassified 237 instances.



**Figure 13:** the distribution of the target variable values (true vs predictions) using soft (left) and hard (right) voting.

## Conclusion

In this project, we address a very serious problem, fake news detection. We utilize the data provided in a Kaggle challenge which contains labeled news articles. We used three embedding techniques TF-IDF, Word2Vec, and Bert to represent our data in the vector space. We trained the three models using all the techniques and compared the results. Finally, we used an ensemble method and compared the performance of hard and soft voting.

# References

[1] Fake News Detection Challenge KDD 2020,  
<https://www.kaggle.com/c/fakenewskdd2020/data>