

UNIVERSITY OF HYDERABAD

&

APPLIED AI

Project

On

Predict Fake News

Content

| | | |
|----------|---|-----------|
| 1 | Introduction | 1 |
| 1.1 | The problem | 1 |
| 1.2 | Data | 2 |
| 1.3 | Key Metrics | 3 |
| 1.4 | Models | 6 |
| 2 | Exploratory data analysis | 7 |
| 2.1 | Data preprocessing | 10 |
| 2.2 | Feature Extraction | 12 |
| 3 | Modeling | 14 |
| 3.1 | Logistic Regression | 14 |
| 3.2 | Naive Bayes | 17 |
| 3.3 | Decision Trees | 18 |
| 3.4 | Random Forest | 21 |
| 3.5 | Xgboost | 23 |
| 3.6 | Summarizing Models | 24 |
| 4 | Error Analysis and Feature Engineering | 26 |
| 4.1 | Logistic Regression | 26 |
| 4.2 | Decision Trees | 28 |
| 4.3 | Random Forest | 30 |
| 4.4 | Bert | 31 |
| 5 | Deployment and Productionization | 33 |
| 5.1 | Flask and templates implementation | 33 |
| 5.2 | Uploading project to github | 37 |
| 5.3 | Deploying project to heroku | 38 |
| 5.4 | Problem faced while deploying | 39 |
| 5.5 | Limitations | 40 |
| 6 | References | 41 |

1. Introduction

1.1 The Problem:

On March 11, 2020, with the drastic increase in COVID-19 cases, the WHO declared COVID -19 a global pandemic which causes panic, anxiety, fear and financial crisis and also led to the rise in false news.

Some of the false news are:

'Lemon juice up the nose will kill coronavirus' , 'Place a halved onion in the corner of your room to catch the COVID-19 germs', 'Sunny weather protects you from COVID-19' , 'The spread of COVID-19 is linked to 5G mobile networks' and many more .

An International Studies showed that India (about 15.94 %), US (about 9.74 %), Brazil (about 8.57 %) and Spain (about 8.03 %) are the four most false news affected countries.

Fake news isn't new problem but due to advancement in technology and increased in social media users, this false news increased sharply in recent years. As these false news now easily be created using bots and diffused via online social networks globally just by a few clicks. Fake news are tend to be created by creators to get viral by attracting their attention and triggering emotional sentiments.

The Covid-19 fake news is just one of the examples of this problem, there are many such problems like fake news based on election or religion sentiments or recent event when twitter's major accounts were hacked and asked for bit coin promising to double their sent amount. The main intention of spreading this fake news is obviously based on political, commercial or achieving other goals.

We should also be concerned in stopping this type of false news spread which leads to violence, panic, division, chaos and hate by using advanced systems. Government has taken various steps in awareness and organising various rules and regulations and also working with high tech companies to overcome this problem. High tech companies are highly invested in technologies which help them finding and identifying various forms of fake news.

So, it is important to make such system which identify whether the news is fake or not and improve our social media so to avoid confusion and handling panic, chaos and violence beforehand.

Purpose

The main purpose is to create a model which identifies whether the news is fake or not and providing warning signs if a news tend to be fake.

These warnings usually encourage people to think critically about any online information and think before sharing or avoiding clicking on links provided in these fake news.

1.2 Data

The source of the dataset is from Kaggle, you can check it [here](#).

Kaggle is a competition platform on which companies post real problem and release their data to data scientist, machine learning engineers (data science community) who solve data science challenges through participating in competition. Kaggle also allows users to search and publish datasets, explore and build models in a web-based data-science environment.

Dataset Overview:

- Data is in file News.csv
- Size of News.csv is 116.86 MB
- News.csv contains 6 columns/features: Unnamed index column, title, text, subject, date, labels
- Number of rows: 44,898

Features:

- **Title:** A headline of a news story.
- **Text:** A descriptive story about the news
- **Subject:** This is one of the categorical feature which has 8 categories of news that are Government News, Middle-east, News, US_News, left-news, politics, politicsNews, worldnews .
- **Date:** following date at which this news has been released.
- **Labels:** It is has two categories: Fake (means fake news) and True (means reliable news or not a fake news)

As the data present in csv format, so, I used pandas library to process the data. I have used pandas.read_csv method to read a comma-separated values (csv) file into dataframe. This method also support iterating or breaking of file into chunks.

The size of the data is 116.86 MB and have 44,898 number of rows which is not very large file, so the processing of file is pretty much fast.

If we need more data we can acquire additional data from other sources also, one of the best source is 'paper with code' where we can find research papers which are very useful for researchers and datasets, libraries, frameworks, models which are very useful in machine learning and data science community. It is open source and self – contained team within Facebook AI research.

We are not using data from paper with code as the dataset is too large and takes more time and efforts to process it and also need strong environment/system setup whereas kaggle has descent amount of data in which we can process and do our task easily with normal system requirement.

1.3 Key Metrics

The main aim of using metrics is to monitor and measure the performance of the model during training and testing.

Confusion Matrix

Confusion matrix is a tabular representation of a classification model performance on the test data which is also very useful in summarizing the prediction of the model. It can be used in both binary classification and multi- class classification. Here, in this case study we will be using binary classification matrix which is comprised of 4 cells and each cell has special name given to it:

- True Positive(TP) : Model correctly predicts the positive class
- False Negative(FN) : Model gives wrong prediction of the negative class
- True Negative(TN) : Model correctly predicts the negative class
- False Positive(FP) : Model gives wrong prediction of the positive class

| | | Actual | |
|-----------|----------|----------------|----------------|
| | | Negative | Positive |
| Predicted | Negative | True Negative | False Negative |
| | Positive | False Positive | True Positive |

Using above details we will define some of the terminologies:

- 1) True Positive Rate(TPR) : It is the ratio of true positive and total actual positive value

$$TPR = \frac{TP}{TP+FN}$$

- 2) True Negative Rate(TNR) : It is the ratio of true negative and total actual negative value

$$TNR = \frac{TN}{TN+FP}$$

- 3) False Positive Rate(FPR) : It is the ratio of false positive and total actual negative value

$$FPR = \frac{FP}{TN+FP}$$

- 4) False Negative Rate(FNR) : It is the ratio of false negative and total actual positive value

$$FNR = \frac{FN}{TP+FN}$$

So, TPR, TNR, FPR and FNR are used in identifying whether the model is good or any dumb model. If TPR and TNR are high and FPR and FNR are low, it is said that model is good. It is not always true it also depends on domain. Confusion matrix can perform even if it has imbalanced dataset.

Precision

It means that out of all the points that model predicted positive, what percentage of them is actually positive. The precision lie between [0,1].

$$\text{Precision} = \frac{TP}{TP+FP}$$

Recall

It means that out of all the positive points in the dataset, how many of them predicted successfully. The recall lie between [0, 1].

$$\text{Recall} = \frac{TP}{TP+FN}$$

F1-Score

If we want both precision and recall to be high, by combining we get F1-score which is harmonic mean of precision and recall. F1-score is less interpretable as compare to precision and recall. It is also ranges between 0 and 1. Higher the F1-score better will be the model.

$$\text{F1-Score} = \frac{2 * (\text{Precision} * \text{Recall})}{(\text{Precision} + \text{Recall})}$$

In case of fake news detection, it is important that True Negative (TN) and True Positive (TP) should be high as actual fake news should be correctly predicted as fake news and actual real news should be predicted as real news.

It is very important that if news is fake, it should not predict it as real news as it will create havoc. So, it means False positive (FP) should be as small as possible and similarly for False negative (FN) that should also be small.

| | | Actual | |
|-----------|-----------|----------------|----------------|
| | | Fake News | Real News |
| Predicted | Fake News | True Negative | False Negative |
| | Real News | False Positive | True Positive |

Same scenario using confusion matrix as in fake news detection, we can see in spam detection in mails and credit card fraud detection.

Accuracy

This is the most commonly used metric for classification. It is a ratio of number of correctly classified points to the total number of points in data. Its value ranges from 0 to 1. In this 0 means model has not predicted any data points correctly and 1 means all the data points correctly classified. It is easily interpreted.

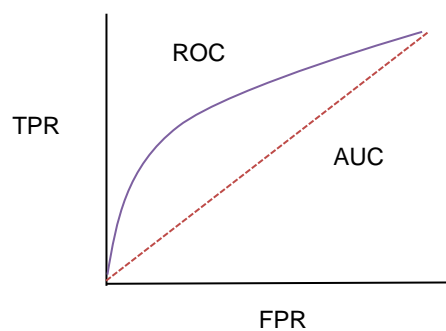
$$\text{Accuracy} = \frac{\text{number of correctly classified points}}{\text{total number of points}} * 100$$

One problem with the accuracy is that it performs very badly for an imbalanced dataset. Although in this case study we can use accuracy as our dataset is balanced.

ROC AND AUC

Receiver Operating Characteristic Curve (ROC) is mainly used for binary classification problem. It is a plot between true positive rate (TPR) and false positive rate (FPR) whose values lie between 0 and 1.

Area under ROC curve is called Area under the curve (AUC) and this also ranges between 0 and 1. Higher the value better will be the model. AUC can easily be impacted by the imbalanced dataset.



Log loss

Log loss is a loss metric which lies between [0, infinite). Lower the loss value better the model. Log loss can be used for both binary classification and multiclass classification. One of the disadvantages of log loss is that it is very difficult to interpret.

Log loss deals with direct probability score unlike the other metrics.

$$\text{Log loss} = - \frac{1}{N} \sum_{i=1}^n \{ (y_i * \log(p_i)) + ((1 - y_i) * \log(1 - p_i)) \}$$

Here,

p_i = Probability of a sample that belongs to class label

y_i = Actual class labels

N = total number of data points

Some of the other metrics that we cannot use in this case study are mean absolute error (MAE), mean squared error (MSE), root mean squared error (RMSE), R-

squared or Coefficient of determination etc. these metrics are used for regression purpose.

1.4 Models

Machine learning model can be categorized into supervised learning, semi-supervised learning, unsupervised learning.

- **Supervised Learning**: In this we have $f(x) = y$, here 'y' supervises in determining the underlying function $f(x)$ means it has labels which help in learning of model.
- **Semi-supervised Learning**: In this we have a majority of the data points without labels and a very few points with the labels.
- **Unsupervised Learning**: In this we don't have any labels in datasets means we do not have y's ($f(x) = y$) so learning is done by determining correlation and relationship by analysing dataset.

Here, we are applying supervised learning and in this learning we have categories as regression and classification from which we are using classification for this case study.

Some of the classification models that we are going to use in this case study:

Naïve Bayes:

This algorithm is based on Bayes theorem. It is a probabilistic based classifier which means it predicts based on probability. It is mainly used for text classification.

Logistic regression:

It is used for predicting the categorical dependent variable using given set of independent variable. It gives the probabilistic values which lie between 0 and 1.

Below models can be used for both classification and regression problems:

Random forest:

This is one of the bagging techniques using the decision trees with large depth as the base models, and combining them is called Random forest.

Bagging Classifier:

It is a High variance, Low bias base models with randomization (both row and column sampling) and aggregation. Decision tree with large depth is one of the examples for this model.

Boosting classifier:

Boosting is Low variance, High bias base models with additive combining (reducing bias while keeping the variance low). Decision tree with shallow depth is one of the examples for this model.

2. Exploratory Data Analysis

In exploratory data analysis, we will analyze and investigate the dataset through which we will discover patterns, anomalies and form hypothesis based on the understanding of the dataset. It basically summarizes the dataset and using visualization tools to visualize the patterns. This visualization helps the stakeholders to better understand the data and take right decision on that basis.

In News.csv, we have 6 features and total rows of 44,898.

At starting we have changed the unnamed column to id column and shuffle the data afterwards as we have seen there are some sequences in Labels column. This is how the data look like:

| | id | title | text | subject | date | Labels |
|---|-------|---|---|--------------|-------------------|--------|
| 0 | 25205 | WATCH: Iowa Woman's Vicious, Racist Assault O... | KCCI reporter Emmy Victor was covering an offi... | News | July 3, 2016 | Fake |
| 1 | 19109 | Hot Mic Catches Someone Saying What We All We... | On Tuesday, a hot mic at a House Republican pr... | News | January 31, 2017 | Fake |
| 2 | 43383 | HOLLYWOOD LIBS Raise Big Money For #CrookedHil... | The Democratic presidential nominee is in the... | left-news | Aug 23, 2016 | Fake |
| 3 | 21631 | OBAMA IS FUNDING Nuclear Weapons That Will Be ... | Meanwhile, Obama has brilliantly distracted Am... | politics | May 25, 2016 | Fake |
| 4 | 20409 | Exclusive: Senator Cruz wants to cap renewable... | NEW YORK (Reuters) - U.S. Senator Ted Cruz wan... | politicsNews | December 15, 2017 | True |
| 5 | 29853 | Donald Trump throws counter punch after Joe Bi... | GENEVA, Ohio (Reuters) - Donald Trump shrugged... | politicsNews | October 28, 2016 | True |
| 6 | 12276 | CNN GETS OWNED! Pundit Explodes On Biased Repo... | The look on their faces when the TRUTH hits th... | politics | Nov 14, 2016 | Fake |
| 7 | 4096 | Trump to nominate ex-NYSE Euronext VP Dawn DeB... | WASHINGTON (Reuters) - U.S. President Donald T... | politicsNews | June 10, 2017 | True |
| 8 | 31895 | Another Prominent Republican Declares: 'I Wil... | Donald Trump s Republican Issue just got a lot... | News | August 8, 2016 | Fake |
| 9 | 7716 | Senate Democrats ask Trump attorney general pi... | WASHINGTON (Reuters) - Nine Democratic senator... | politicsNews | January 17, 2017 | True |

Fig: Dataset

The first thing is that we need to check our dataset is balanced or not on the basis of labels. So, we visualized that we have 23,481 Fake news and 21,417 True/Real news which means our data is balanced.

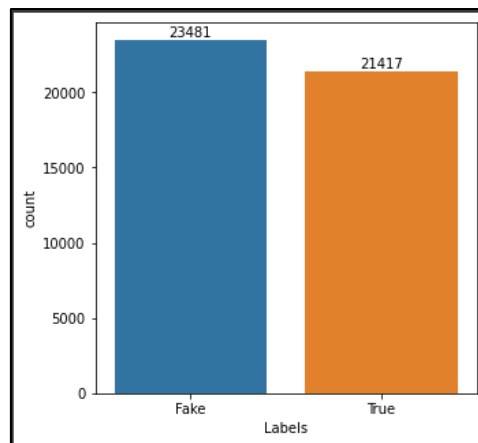


Fig: Labels data visualization

After analyzing Labels and behavior, now we will understand about feature subject. We have total 8 categories in subject : politicsNews, worldnews, News, politics, left-news, Government News, US_News, Middle-east.

Key Observation

- Most of the news is from subject- politicsNews and worldnews.
- Least number of news is from US_News and Middle-east.

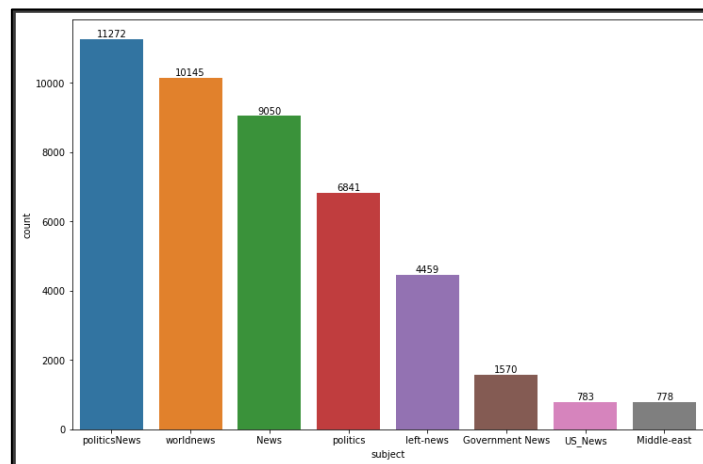


Fig: Visualizing news count based on subjects

Now, we need to know how many true and fake news does each subject has.

Key Observation

- It can be seen that all the news from politicsNews and worldnews having only true news that means these news are from authentic source.
- All the news from News, politics, left-news, middle-east, Government News, US_News having only Fake news that means these are not from authentic source.

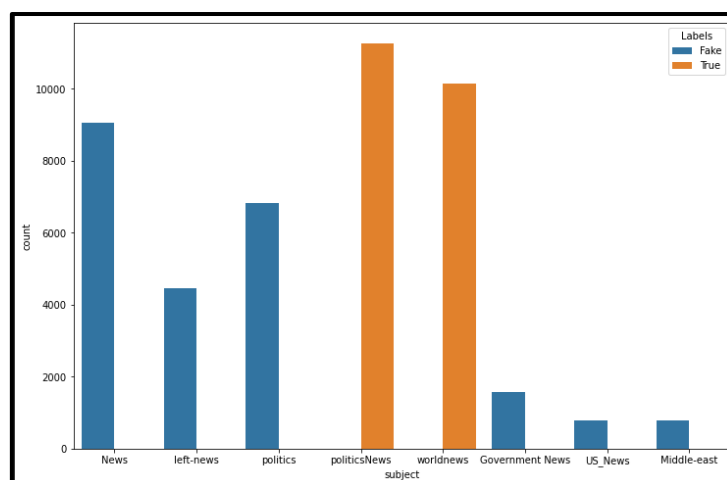


Fig: Visualizing news subjects based on labels

Now, we will apply word cloud using our dataset .Word cloud is a graphical representation of words frequency, those words will be highlighted with different colors and fonts based on frequency and exclude common words or stop words. Below are the word clouds for both true and fake news.

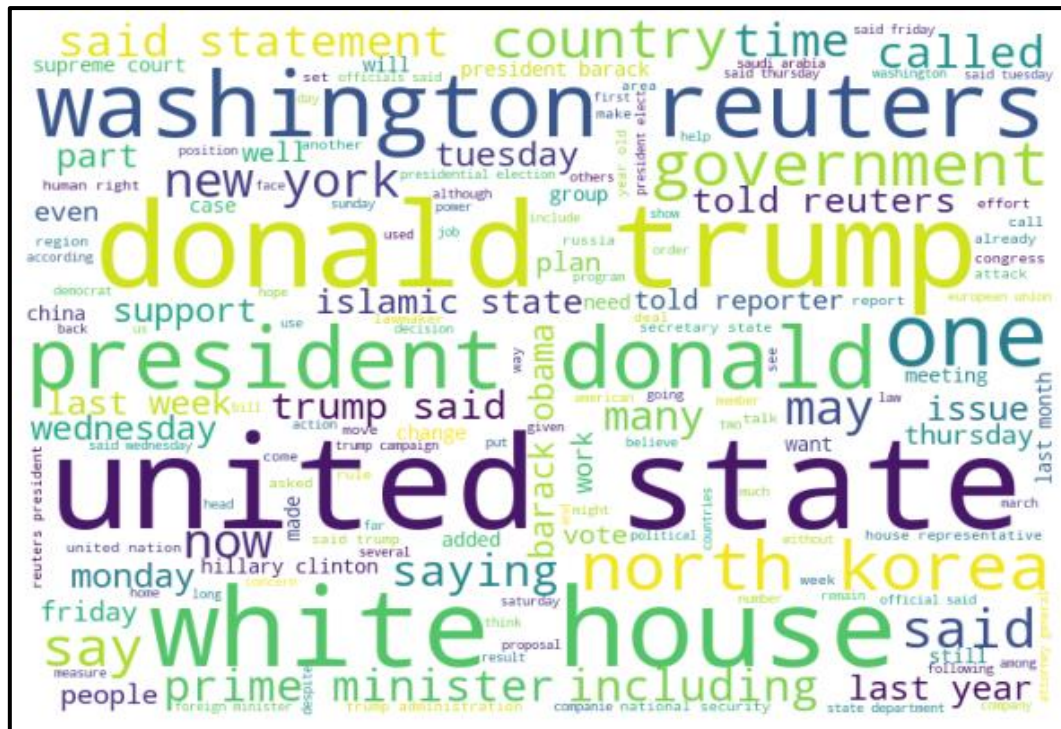


Fig: Word cloud for true news

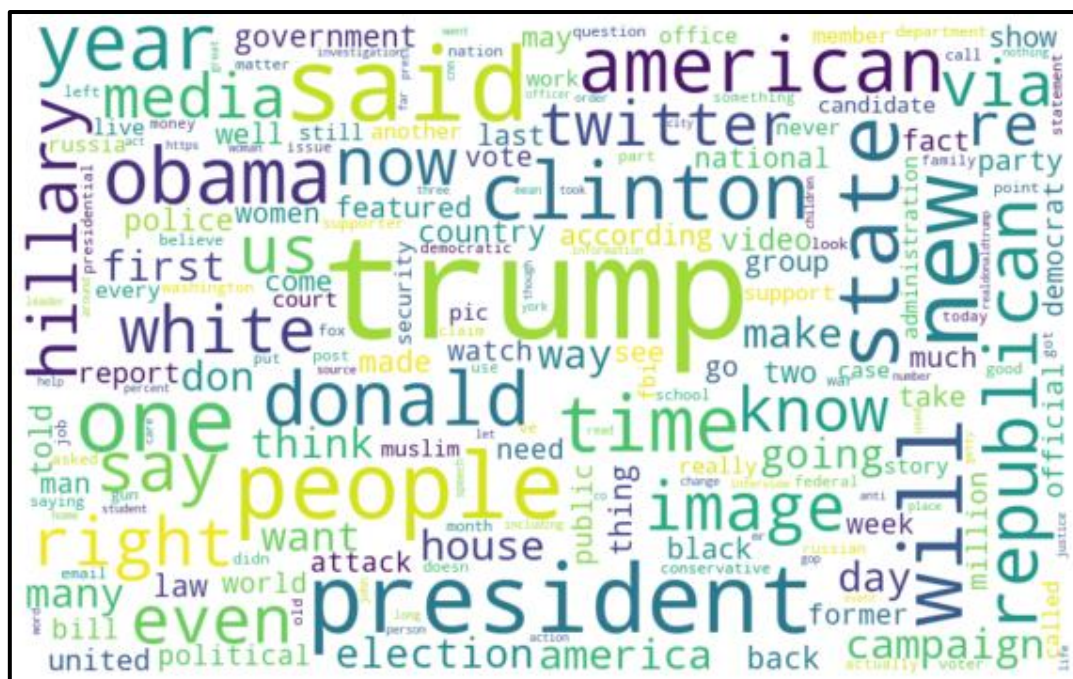


Fig: Word cloud for fake news

2.1 Data Preprocessing

We will be taking both title and text feature for pre-processing. In this method we will be cleaning our text data so that it can be fed to the model. It is important as it will help in good textual analysis and making better accurate decision.

- **Checking missing values** : we have checked for missing values but we don't have any in any of the feature.
- **Deduplication** : As we have to deduplication as these data take more spaces. Although we checked and didn't get duplicates.

Following steps we have taken to preprocess our text data:

- **Removing urls**
- **Removing numbers from text**
- **Decontracting words**: Some words are written in contracted forms which are shortened version of words. They are created by removing one of the vowels from the word. Example: do not -> don't .
So, expanding contracted words to its original form helps in text standardization.
- **Removing punctuations and special characters** : It helps in get rid of noise so that we can have better text analysis.
- **Tokenization** : In this we split the sentence into list of words to perform other operations
- **Making text in lowercase**: It is important to make all the words in lower case as it is also compared with stopwords .As words which are stopwords and are not in lower case will not be removed.
- **Removing stopwords**: these words have low –level information and are most common words that has no significance in constructing meaningful features from text i.e. why these words are excluded and give more importance to high-level information.
- **Lemmatization**: It takes consideration to the related forms of the words means to determine the lemma (i.e. root form) of a word based on intended meaning. Example: lemma form of studies -> study

After applying all the steps to the data we will got a preprocessed text and will store in a pickle file so, we don't have to run again and again and can load data from pickle file and do further analysis.

Let us now get more details about data by trying to find words that are more frequent in both fake news and true news.

We can see words like trump, said, president are appearing more number of times in true news and words like said giving more importance in fake news. As these news has more news regarding election and at the same time we have most fake news travelling i.e. why we are seeing more words related to elections.

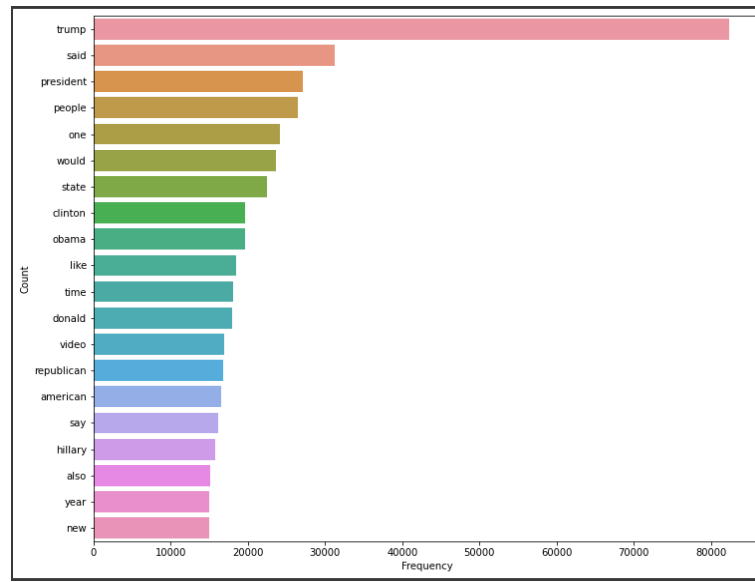


Fig: Visualizing word frequency for True news

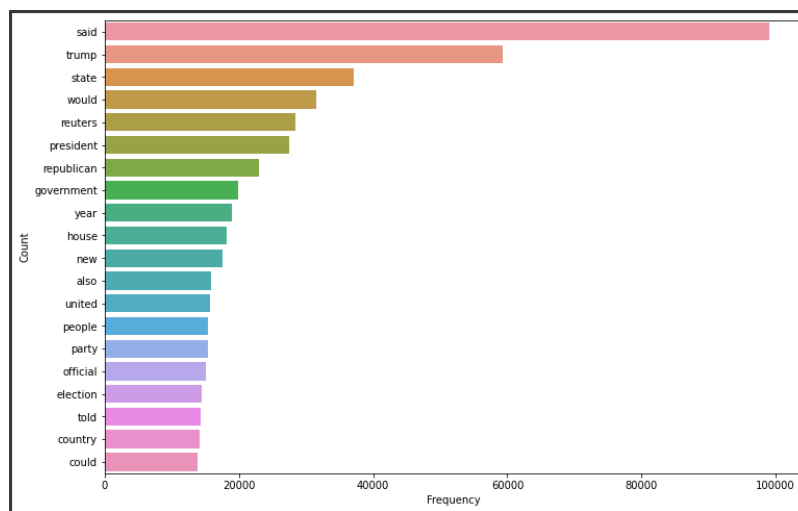


Fig: Visualizing word frequency for Fake news

LABELS CONVERTS TO NUMERICAL FORM

We have labels in text form so, it is important to convert them in numerical form. So, we taken LabelEncoder() which will provide binary representation to our data i.e. for fake label it is 0 and for true label it is 1.

2.2 Feature Extraction

Featurization is about converting the data of one type into a numerical vector form. Whenever we apply any algorithm in NLP, it works on numerical values so; we cannot directly feed the text data into the algorithm.

For this we have various feature extraction techniques that we are using in this case:

Bag of Words

It is a technique to convert text data into vector form by constructing a 'd' dimensional vector for each document and filling values based on number of times the corresponding word occur in a given document. Here, each word considered as different dimensions.

The result of Bag of Words is always a sparse vector means most the dimensions in it have 0 as value.

Example: Here D1, D2, D3, D4 are documents

D1: this pasta is very tasty and affordable

D2: this pasta is not tasty and is affordable

D3: this pasta is delicious and cheap

D4: pasta is tasty and pasta tastes good

Step 1: It will create a dictionary of set of words means all unique words.

Step 2: It will create a vector of d dimension based on number of unique words in a corpus.

| | | | | | | | | | | | |
|------|-------|----|------|-------|-----|------------|-----|-----------|-------|--------|------|
| this | pasta | is | very | tasty | and | affordable | not | delicious | cheap | tastes | good |
| | | | | | | | | | | | |

Step 3: It will create a vector for each document (V1, V2, V3 and V4) with filled values based on number of times the corresponding word occur.

| | | | | | | | | | | | | |
|-----|------|-------|----|------|-------|-----|------------|-----|-----------|-------|--------|------|
| v1: | this | pasta | is | very | tasty | and | affordable | not | delicious | cheap | tastes | good |
| | 1 | 1 | 1 | 1 | 1 | 1 | | 1 | 0 | 0 | 0 | 0 |

| | | | | | | | | | | | | |
|-----|------|-------|----|------|-------|-----|------------|-----|-----------|-------|--------|------|
| v2: | this | pasta | is | very | tasty | and | affordable | not | delicious | cheap | tastes | good |
| | 1 | 1 | 2 | 0 | 1 | 1 | | 1 | 1 | 0 | 0 | 0 |

Similarly, it will create for V3 and V4.

If we want to calculate difference between two vectors

$$\text{Length}(V1, V2) = ||V1 - V2||$$

TF-IDF(Term Frequency – Inverse Document Frequency)

It is a statistical measure that evaluates how relevant a word is to a document in a collection of document. This is done by multiplying both term frequency (means how many times a word appears in a document) and inverse document frequency (means idf of the word across a set of document).

$$\text{TF-IDF}(wi, dj) = \text{TF}(wi, dj) * \text{IDF}(wi, Dc)$$

Where wi = Number of words,

dj = Number of documents,

Dc = Data corpus

TF(Term Frequency): TF of any word in any document is how often does words occur in document. Its value lie between 0 and 1 means can be helpful in interpreting probability also.

$$\text{TF}(wi, dj) = \frac{(\text{Number of times 'wi' occurs in 'dj'})}{(\text{Total number of words in 'dj'})}$$

IDF(Inverse Document Frequency): IDF of a word in a given data corpus is given by logarithmic ratio of total number of document in a corpus to the total number of document which contain that word.

$$\text{IDF}(wi, Dc) = \log\left(\frac{\text{Total number of Documents(N)}}{\text{Total number of documents containing 'wi'}}$$

TF will be high when ' wi ' is more frequent in ' dj ' and IDF is high when ' wi ' is rare in ' Dc '.

BOW and TF-IDF

- Both don't take the semantic meaning into consideration like (cheap, affordable). This is one of the drawbacks.
- The main difference between BOW and TF-IDF is that instead of using frequency counts as the values in d-dimensional vector for each word as in BOW, we use TF-IDF score for the words in TF-IDF vector representation.
- In our case study, for BOW we are taking CountVectorizer() and for TF-IDF we are taking TfidfVectorizer() for feature extraction.
- For both of them we are taking 5000 feature as it is a sparse vector need large feature

DOC2VEC

It is a model that represent each document as a vector. We are using DBOW (Distributed Bag of Words) which are obtained by training a neural network on the synthetic task of predicting a target word just from the full document's doc-vector

As per our case we first tagged each document with unique number and its type means whether it is belong to train or test. Now we will combine both test and train data and apply the Doc2Vec algorithm.

After getting vocabulary we will train the model. Now, split train and test data based on tagged and converting to vector form and getting dense vector of 300 dimensions.

The main advantage of taking doc2vec is that it takes semantic meaning into consideration. Vectors are in dense form.

Now, we have vectorize data from Bow, Tfidf and doc2vec and these are implemented in various machine learning models and check accuracy which vectorizer work better as per given data.

3. Modeling

3.1 Logistic Regression

It is used for predicting the categorical dependent variable using given set of independent variable. It gives the probabilistic values which lie between 0 and 1. It can be used for both binary and multiclass classification problem. The objective of logistic regression is to find a plane ' π ' with an optimal ' w ' that could maximize $\sum_{i=1}^n y_i * w^T \cdot x_i$

Where, $w^T \cdot x_i$ = distance from ' x_i ' to the plane ' π ', $y_i \in \{-1, 1\}$

We have implemented logistic regression in two way – one way is without hyperparameter tuning and another with hyperparameter tuning.

Without hyperparameter – tuning

We have 3 types of vectorizer i.e. countvectorizer(Bag of word),tf-idf and doc2vec .

```
Accuracy: 0.9915367483296214
precision    recall  f1-score   support

      0       0.99      0.99      0.99     5893
      1       0.99      0.99      0.99     5332

 accuracy          0.99      0.99      0.99     11225
  macro avg       0.99      0.99      0.99     11225
 weighted avg     0.99      0.99      0.99     11225

***** LOG-LOSS *****
log loss on train dataset : 0.012357445214475826
log loss on test dataset  : 0.04015607349075472
```

Fig: Using Countvectorizer

```
Accuracy: 0.9832516703786192
precision    recall  f1-score   support

      0       0.99      0.98      0.98     5893
      1       0.98      0.99      0.98     5332

 accuracy          0.98      0.98      0.98     11225
  macro avg       0.98      0.98      0.98     11225
 weighted avg     0.98      0.98      0.98     11225

***** LOG-LOSS *****
log loss on train dataset : 0.06252689586795193
log loss on test dataset  : 0.0682573725970566
```

Fig: Using Tf-idfvectorizer


```

Accuracy: 0.990913140311804
      precision    recall  f1-score   support

     0       0.99      0.99      0.99     5893
     1       0.99      0.99      0.99     5332

 accuracy          0.99          0.99          0.99     11225
  macro avg       0.99          0.99          0.99     11225
 weighted avg     0.99          0.99          0.99     11225

***** LOG-LOSS *****
log loss on train dataset : 0.016763562737708557
log loss on test dataset : 0.02771546920655845

```

Fig: Using Doc2vec

We can see in above figures, countvectorizer is having maximum accuracy(99.15%) in comparison with other vectorizer and even the log loss is also very less.

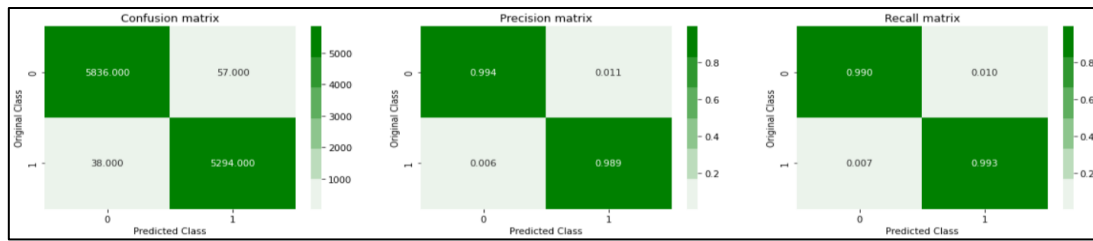


Fig: Confusion matrix for countvectorizer

With hyperparameter – tuning

Here, we have taken parameter 'C' which is inverse of regularization and do hyperparameter tuning.

```

Best parameters
gridsearch: {'C': 0.1}

Accuracy: 0.992694877505568
The train log loss is: 0.07684809980502229
The test log loss is: 0.08347172118220665

```

Fig: For countvectorizer

```

Best parameters
gridsearch: {'C': 10}

Accuracy: 0.9890423162583519
The train log loss is: 0.028768972766878485
The test log loss is: 0.037262300901035765

```

Fig: For tf-idfvectorizer

```

Best parameters
gridsearch: {'C': 0.1}

Accuracy: 0.991358574610245
The train log loss is: 0.018944370463506843
The test log loss is: 0.026115717136598514

```

Fig: For Doc2vec

We can see in above figure, maximum accuracy is of countvectorizer i.e 99.26%. So, we will be taking its confusion matrix in comparison to others.

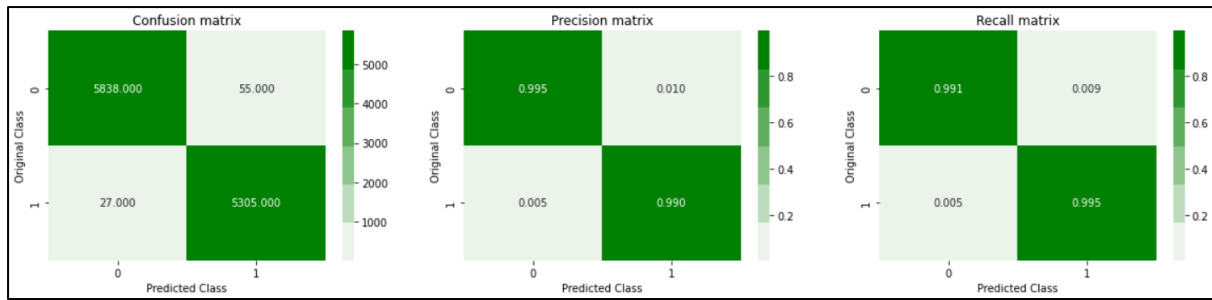


Fig: Confusion matrix for countvectorizer

Key Observation:

- For counvectorizer, we got best value for 'C' as 0.1 due to this accuracy is 99.26 % which is increased from the previous one.
- From confusion matrix we can see 5305 points are correctly classified as true news whereas we have error of 27 points which are misclassified as fake news. Also for 5838 points are correctly classified as fake news but 55 of them are misclassified as true news.
- If we compare before hyperparameter tuning we can see the improvement, misclassified datapoints was 57 and 38 which improved to 55 and 27.

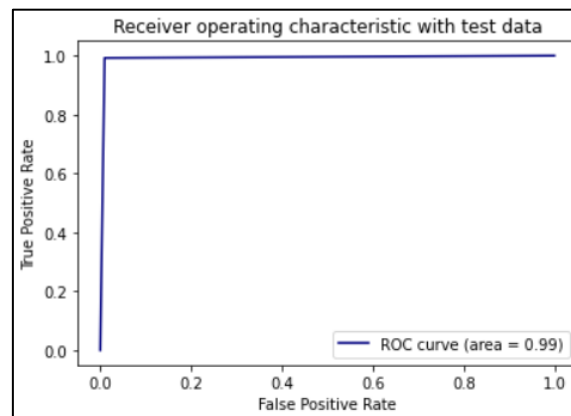


Fig: ROC curve

ROC(Receiver operating characteristics) graphs provide a simple way to summarize all of the information. In y-axis we have true positive rate which is same as sensitivity which tells what proportion of true news samples were correctly classified and in x-axis we have false positive rate which is same as (1- specificity) which tells the proportion of fake news samples that were incorrectly classified and are false positive. In this graph showing an area of 0.99 is covered.

Here, in Logistic Regression we have best result of 99.26% of accuracy and much better precision and recall of countvectorizer which is hyperparameter tuned.

3.2 Naive Bayes

This algorithm is based on Bayes theorem. It is a probabilistic based classifier which means it predicts based on probability. It is mainly used for text classification. Time complexity of naive bayes is $O(d*c)$, where d is query vector's dimension and c is total classes.

We can clearly see that accuracy of tfidfvectorizer i.e. 93.10 % which is a lot better than countvectorizer and near to doc2vec. So, taking confusion matrix of tfidfvectorizer which has maximum accuracy.

```

Accuracy: 0.8824053452115813
      precision    recall  f1-score   support

     0       0.96      0.81      0.88     5893
     1       0.82      0.96      0.89     5332

 accuracy          0.88     11225
 macro avg       0.89      0.89      0.88     11225
 weighted avg    0.89      0.88      0.88     11225

***** LOG-LOSS *****
log loss on train dataset : 2.4884923028364647
log loss on test dataset : 2.650564392957732

```

Fig: Using countvectorizer

```

Accuracy: 0.9310467706013363
      precision    recall  f1-score   support

     0       0.96      0.91      0.93     5893
     1       0.90      0.96      0.93     5332

 accuracy          0.93     11225
 macro avg       0.93      0.93      0.93     11225
 weighted avg    0.93      0.93      0.93     11225

***** LOG-LOSS *****
log loss on train dataset : 1.8402621861700827
log loss on test dataset : 1.9463324331426142

```

Fig: Using tf-idfvectorizer

```

Accuracy: 0.9294432071269487
      precision    recall  f1-score   support

     0       0.94      0.92      0.93     5893
     1       0.92      0.94      0.93     5332

 accuracy          0.93     11225
 macro avg       0.93      0.93      0.93     11225
 weighted avg    0.93      0.93      0.93     11225

***** LOG-LOSS *****
log loss on train dataset : 0.2834765729396397
log loss on test dataset : 0.2871379280759485

```

Fig: Using Doc2vec

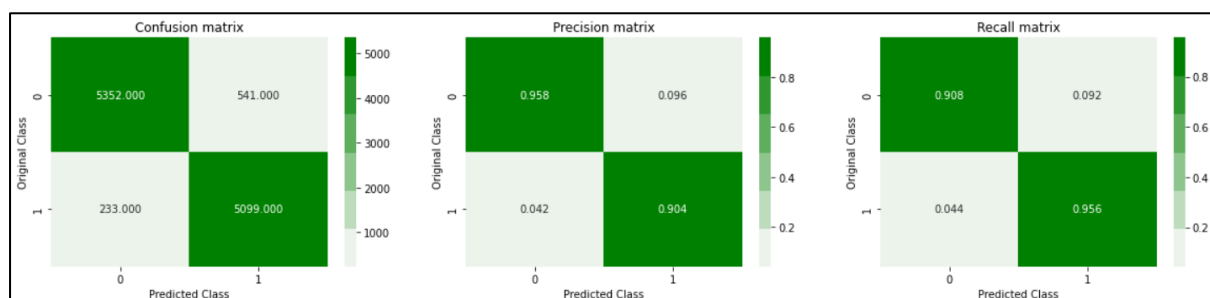


Fig: Confusion matrix for tf-idfvectorizer

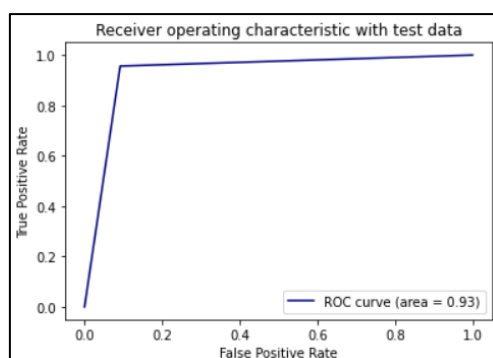


Fig: ROC curve

Key Observation

- From confusion matrix we can see 5099 datapoints are correctly classified as true news whereas we have error of 233 datapoints which are misclassified as fake news .Also for 5352 datapoints are correctly classified as fake news but 541 of them are misclassified as true news.
- As compare to logistic model this model is performing much worse.
- Also the ROC curve area is about 0.93
- Also we can see that log loss also increased for this model.

Here, in Naive bayes we have best result of 93.10% of accuracy of tf-idfvectorizer and much worse classification as compared to logistic regression.

3.3 Decision Trees

Decision trees basically behave like a nested if – else classifier which follows if-else rule. We can convert our nested if-else condition into a tree form. So at each node of the tree we are essentially checking a condition similar to if checks and based on the result of this checking we are deciding which path to take. Decision trees are highly interpretable. Also interpretability goes down as the depth of the tree increases.

Without hyperparameter – tuning

We can clearly see that tf-idfvectorizer accuracy excel among other vectorizer which is 99.42 %.So, we will be taking confusion matrix of tf-idfvectorizer to visualize.

| | | | | |
|---|-----------|--------|----------|---------|
| Accuracy: 0.9935857461024499 | | | | |
| | precision | recall | f1-score | support |
| 0 | 0.99 | 0.99 | 0.99 | 5893 |
| 1 | 0.99 | 0.99 | 0.99 | 5332 |
| accuracy | | | 0.99 | 11225 |
| macro avg | 0.99 | 0.99 | 0.99 | 11225 |
| weighted avg | 0.99 | 0.99 | 0.99 | 11225 |
| ***** LOG-LOSS ***** | | | | |
| log loss on train dataset : 9.992007221626413e-16 | | | | |
| log loss on test dataset : 0.22154048110766858 | | | | |

Fig: Using countvectorizer

| | | | | |
|---|-----------|--------|----------|---------|
| Accuracy: 0.9942984409799555 | | | | |
| | precision | recall | f1-score | support |
| 0 | 0.99 | 1.00 | 0.99 | 5893 |
| 1 | 0.99 | 0.99 | 0.99 | 5332 |
| accuracy | | | 0.99 | 11225 |
| macro avg | 0.99 | 0.99 | 0.99 | 11225 |
| weighted avg | 0.99 | 0.99 | 0.99 | 11225 |
| ***** LOG-LOSS ***** | | | | |
| log loss on train dataset : 9.992007221626413e-16 | | | | |
| log loss on test dataset : 0.19692487209570553 | | | | |

Fig: Using tf-idfvectorizer

```

Accuracy: 0.7901113585746102
      precision    recall  f1-score   support

     0       0.80      0.81      0.80      5893
     1       0.78      0.77      0.78      5332

 accuracy
macro avg      0.79      0.79      0.79      11225
weighted avg    0.79      0.79      0.79      11225

***** LOG-LOSS *****
log loss on train dataset : 9.992007221626413e-16
log loss on test dataset : 7.249296854023124

```

Fig: Using Doc2vec

In confusion matrix we can clearly see that true news which misclassified as fake news are 36 and fake news which misclassified as true news are 28.

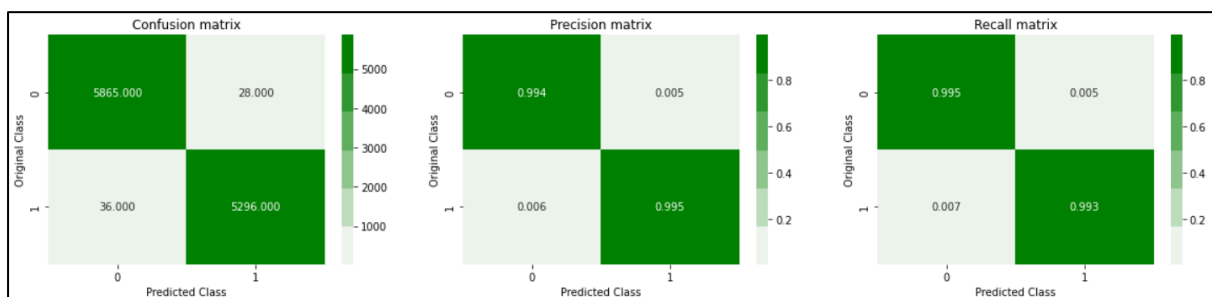


Fig: Confusion matrix for tf-idfvectorizer

With hyperparameter – tuning

Here, after hyperparameter tuning we got accuracy of 99.33% which is same for both tf-idfvectorizer and countvectorizer and way much better than doc2vec. As loss is much less in tf-idf so we will be picking tf-idfvectorizer in this case which has best parameter for criterion as entropy, for max_depth we have 3 and for min_samples_leaf we have 1.

```

Best parameters
gridsearch: {'criterion': 'entropy', 'max_depth': 3, 'min_samples_leaf': 1}

Accuracy: 0.9933184855233853
The train log loss is: 0.03177411284961731
The test log loss is: 0.03781291371916189

      precision    recall  f1-score   support

     0       1.00      0.99      0.99      5893
     1       0.99      1.00      0.99      5332

 accuracy
macro avg      0.99      0.99      0.99      11225
weighted avg    0.99      0.99      0.99      11225

```

Fig: Using countvectorizer

```

Best parameters
gridsearch: {'criterion': 'entropy', 'max_depth': 3, 'min_samples_leaf': 1}

Accuracy: 0.9933184855233853
The train log loss is: 0.030172064570203272
The test log loss is: 0.03477697712120355

      precision    recall  f1-score   support

     0       1.00      0.99      0.99      5893
     1       0.99      1.00      0.99      5332

 accuracy
macro avg      0.99      0.99      0.99      11225
weighted avg    0.99      0.99      0.99      11225

```

Fig: Using tf-idfvectorizer

```

Best parameters
gridsearch: {'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 6}

Accuracy: 0.8721603563474387
The train log loss is: 0.27373796592567545
The test log loss is: 0.4115067257943339


```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.87 | 0.89 | 0.88 | 5893 |
| 1 | 0.88 | 0.85 | 0.86 | 5332 |
| accuracy | | | 0.87 | 11225 |
| macro avg | 0.87 | 0.87 | 0.87 | 11225 |
| weighted avg | 0.87 | 0.87 | 0.87 | 11225 |

Fig: Using Doc2vec

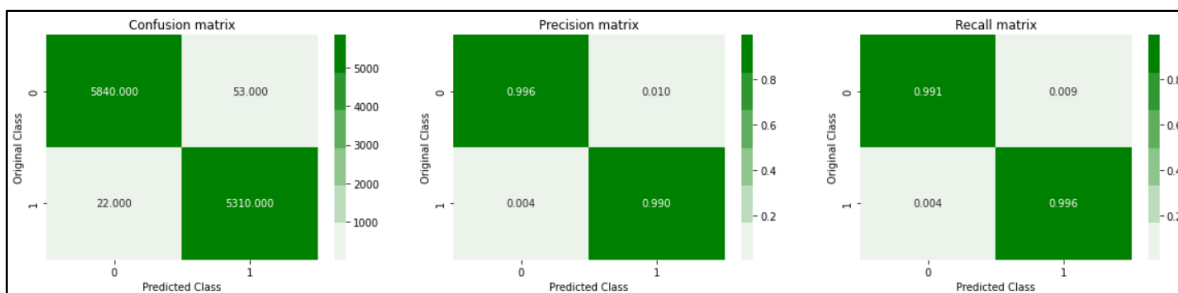


Fig: Confusion matrix for tf-idfvectorizer

Key Observation:

- In confusion matrix, we can see that true news which are correctly classified are 5310 data points whereas have some error in classifying some of the true news and classified as fake news i.e. 22 data points.
- For fake news it is able to classify 5840 data points correctly and misclassified 53 data points as true news.
- As we can see before hyperparameter tuning we have better accuracy but the loss increase much more than others so after hyperparameter tuning we can see that our accuracy slightly decreases but improves the loss much better.
- Here, Roc curve is giving almost a perfect classifier with an area of 0.99.

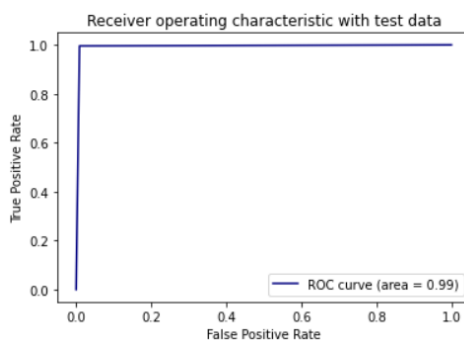


Fig: ROC curve

Here, in decision tree we got best accuracy of about 99.3%.

3.4 Random Forest

Bagging is a High variance, Low bias base models with randomization (both row and column sampling) and aggregation. Decision tree with large depth as the base model and combining them we get random forest which is one of the examples of bagging technique. When we have multi-core we can build each learner in each core i.e. why it is called as trivially parallelizable

Without hyperparameter – tuning

We can clearly see that countvectorizer accuracy excel among other vectorizer which is 99.67 %. So, we will be taking confusion matrix of countvectorizer to visualize. In confusion matrix we can see an excellent result till now as we only have misclassified data points of true news are 14 and misclassified data points of fake news are 22.

```
Accuracy: 0.9967928730511225
      precision    recall  f1-score   support

     0       1.00      1.00      1.00     5893
     1       1.00      1.00      1.00     5332

 accuracy
macro avg       1.00      1.00      1.00     11225
weighted avg     1.00      1.00      1.00     11225

***** LOG-LOSS *****
log loss on train dataset : 0.018508451771410993
log loss on test dataset  : 0.058241186569150165
```

Fig: Using countvectorizer

```
Accuracy: 0.9965256124721603
      precision    recall  f1-score   support

     0       1.00      1.00      1.00     5893
     1       1.00      1.00      1.00     5332

 accuracy
macro avg       1.00      1.00      1.00     11225
weighted avg     1.00      1.00      1.00     11225

***** LOG-LOSS *****
log loss on train dataset : 0.018881774237634392
log loss on test dataset  : 0.055666124700710626
```

Fig: Using tf-idfvectorizer

```
Accuracy: 0.9591982182628063
      precision    recall  f1-score   support

     0       0.95      0.98      0.96     5893
     1       0.97      0.94      0.96     5332

 accuracy
macro avg       0.96      0.96      0.96     11225
weighted avg     0.96      0.96      0.96     11225

***** LOG-LOSS *****
log loss on train dataset : 0.10278388040500866
log loss on test dataset  : 0.3171367452747571
```

Fig: Using Doc2vec

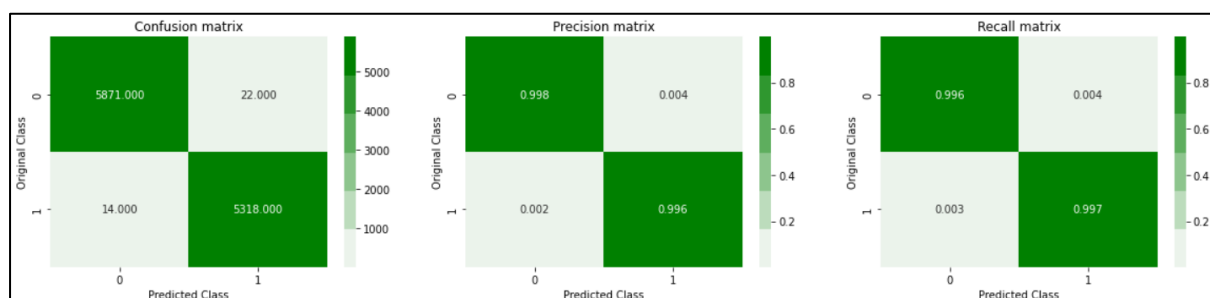


Fig: Confusion matrix for countvectorizer

With hyperparameter – tuning

Even after doing the hyperparameter tuning we got the best accuracy as 99.67% with parameter max_depth as 30, min_sample_leaf as 1, min_samples_split as 2 , n_estimators as 100.

```
Best parameters
gridsearch: {'max_depth': 30, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}

Accuracy: 0.996792873051225
The train log loss is: 0.0017934620436313473
The test log loss is: 0.014592317281351574
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 1.00 | 1.00 | 1.00 | 5893 |
| 1 | 1.00 | 1.00 | 1.00 | 5332 |
| accuracy | | | 1.00 | 11225 |
| macro avg | 1.00 | 1.00 | 1.00 | 11225 |
| weighted avg | 1.00 | 1.00 | 1.00 | 11225 |

Fig: Using countvectorizer

```
Best parameters
gridsearch: {'max_depth': 30, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}

Accuracy: 0.9966146993318485
The train log loss is: 0.0017984169263807277
The test log loss is: 0.015282449264014591
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 1.00 | 1.00 | 1.00 | 5893 |
| 1 | 1.00 | 1.00 | 1.00 | 5332 |
| accuracy | | | 1.00 | 11225 |
| macro avg | 1.00 | 1.00 | 1.00 | 11225 |
| weighted avg | 1.00 | 1.00 | 1.00 | 11225 |

Fig: Using tf-idfvectorizer

```
Best parameters
gridsearch: {'max_depth': 30, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}

Accuracy: 0.9689086859688196
The train log loss is: 0.013117168482155477
The test log loss is: 0.09325074977491894
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.97 | 0.97 | 0.97 | 5893 |
| 1 | 0.97 | 0.97 | 0.97 | 5332 |
| accuracy | | | 0.97 | 11225 |
| macro avg | 0.97 | 0.97 | 0.97 | 11225 |
| weighted avg | 0.97 | 0.97 | 0.97 | 11225 |

Fig: Using Doc2vec

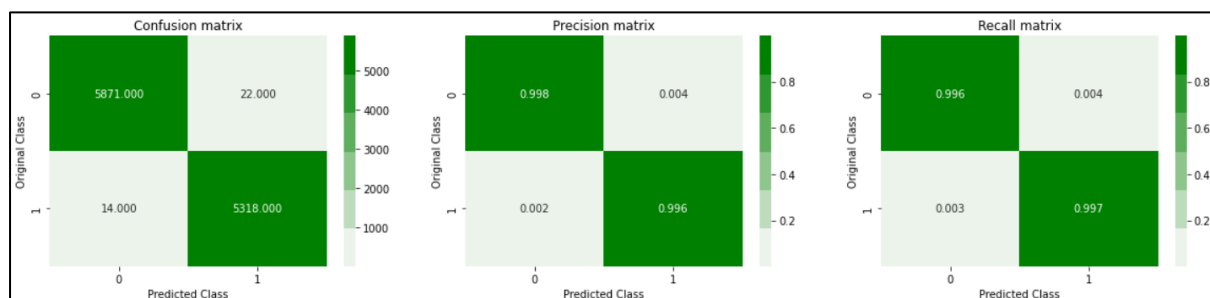


Fig: Confusion matrix for countvectorizer

Key Observation

- In confusion matrix, we can clearly see that we got an excellent result till now as true news classified correctly as 5318 and misclassified only 14 data points as fake news
- Also for fake news correctly classified 5871 data points and misclassified only 22 data points as true news.
- This classifier can be said as perfect classifier as its ROC curve area is about 1

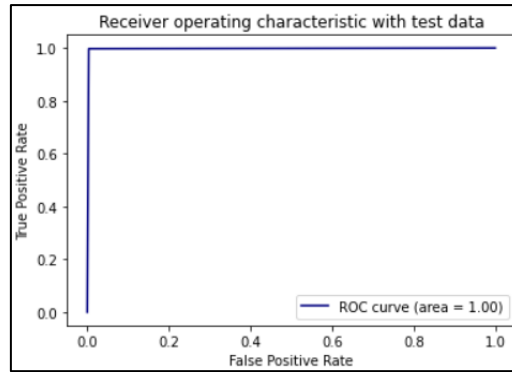


Fig: ROC curve

Here, in random forest we got an best accuracy of 99.67% which is best till now as compare to other models.

3.5 Xgboost

It is a decision tree based ensemble that uses a gradient boosting framework which uses low variance ,high bias base models with additive combining(reducing bias while keeping the variance low). This algorithm follows parallelization, tree pruning for system optimization and for algorithm enhancement we can use regularization, sparsity, cross validation.

We can clearly see that countvectorizer and tfidfvectorizer has accuracy of 99.51% which is same only have different log loss but much better than doc2vec. So, we will be taking confusion matrix of countvectorizer to visualize(can take any between countvectorizer and tfidfvectorizer).

```
Accuracy: 0.9951002227171493
      precision    recall  f1-score   support

     0       1.00      0.99      1.00     5893
     1       0.99      1.00      0.99     5332

   accuracy          0.99      1.00      1.00     11225
  macro avg          0.99      1.00      1.00     11225
 weighted avg          1.00      1.00      1.00     11225

***** LOG-LOSS *****
log loss on train dataset : 0.011687293422820027
log loss on test dataset : 0.0200048320804224
```

Fig: Using countvectorizer

```
Accuracy: 0.9951002227171493
      precision    recall  f1-score   support

     0       1.00      0.99      1.00     5893
     1       0.99      1.00      0.99     5332

   accuracy          0.99      1.00      1.00     11225
  macro avg          0.99      1.00      1.00     11225
 weighted avg          0.99      1.00      1.00     11225

***** LOG-LOSS *****
log loss on train dataset : 0.010263292384170761
log loss on test dataset : 0.01991628604236161
```

Fig: Using tf-idfvectorizer

```
Accuracy: 0.9472605790645879
      precision    recall  f1-score   support

     0       0.95      0.95      0.95     5893
     1       0.94      0.95      0.94     5332

   accuracy          0.95      0.95      0.95     11225
  macro avg          0.95      0.95      0.95     11225
 weighted avg          0.95      0.95      0.95     11225

***** LOG-LOSS *****
log loss on train dataset : 0.1994240501233916
log loss on test dataset : 0.22162619754236465
```

Fig: Using Doc2vec

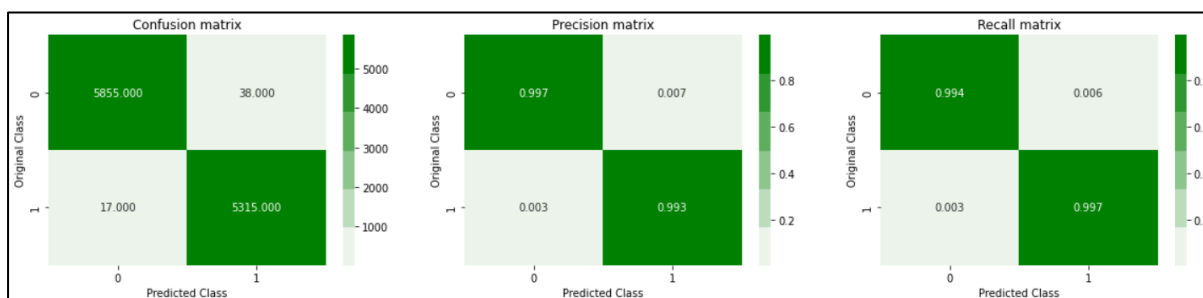


Fig: Confusion matrix for tf-idfvectorizer

Key Observation:

- In confusion matrix, we can see that true news which are correctly classified are 5315 data points whereas have some error in classifying some of the true news and classified as fake news i.e. 17 data points.
- For fake news it is able to classify 5855 data points correctly and misclassified 38 data points as true news.

Here, Xgboost give the best result with an accuracy of 99.51%

3.6 Summarizing Models

Here, we can see accuracy of each model based on given technique and given hyperparameter tuning

| index ▲ | Model | Technique | Hyperparameter-Tuning | Accuracy |
|---------|---------------------|--------------|-----------------------|--------------------|
| 0 | Logestic Regression | BAG OF WORDS | No | 0.9915367483296214 |
| 1 | Logestic Regression | TF-IDF | No | 0.9832516703786192 |
| 2 | Logestic Regression | DOC2VEC | No | 0.990913140311804 |
| 3 | Logestic Regression | BAG OF WORDS | Yes | 0.992694877505568 |
| 4 | Logestic Regression | TF-IDF | Yes | 0.9890423162583519 |
| 5 | Logestic Regression | DOC2VEC | Yes | 0.991358574610245 |
| 6 | Naive Bayes | BAG OF WORDS | No | 0.8824053452115813 |
| 7 | Naive Bayes | TF-IDF | No | 0.9310467706013363 |
| 8 | Naive Bayes | DOC2VEC | No | 0.9294432071269487 |
| 9 | Decision Tree | BAG OF WORDS | No | 0.9935857461024499 |
| 10 | Decision Tree | TF-IDF | No | 0.9942984409799555 |
| 11 | Decision Tree | DOC2VEC | No | 0.7901113585746102 |
| 12 | Decision Tree | BAG OF WORDS | Yes | 0.9933184855233853 |
| 13 | Decision Tree | TF-IDF | Yes | 0.9933184855233853 |
| 14 | Decision Tree | DOC2VEC | Yes | 0.8721603563474387 |
| 15 | Random Forest | BAG OF WORDS | No | 0.996792873051225 |
| 16 | Random Forest | TF-IDF | No | 0.9965256124721603 |
| 17 | Random Forest | DOC2VEC | No | 0.9591982182628063 |
| 18 | Random Forest | BAG OF WORDS | Yes | 0.996792873051225 |
| 19 | Random Forest | TF-IDF | Yes | 0.9966146993318485 |
| 20 | Random Forest | DOC2VEC | Yes | 0.9689086859688196 |
| 21 | Xgboost | BAG OF WORDS | No | 0.9951002227171493 |
| 22 | Xgboost | TF-IDF | No | 0.9951002227171493 |
| 23 | Xgboost | DOC2VEC | No | 0.9472605790845879 |

Fig: Summarizing all model

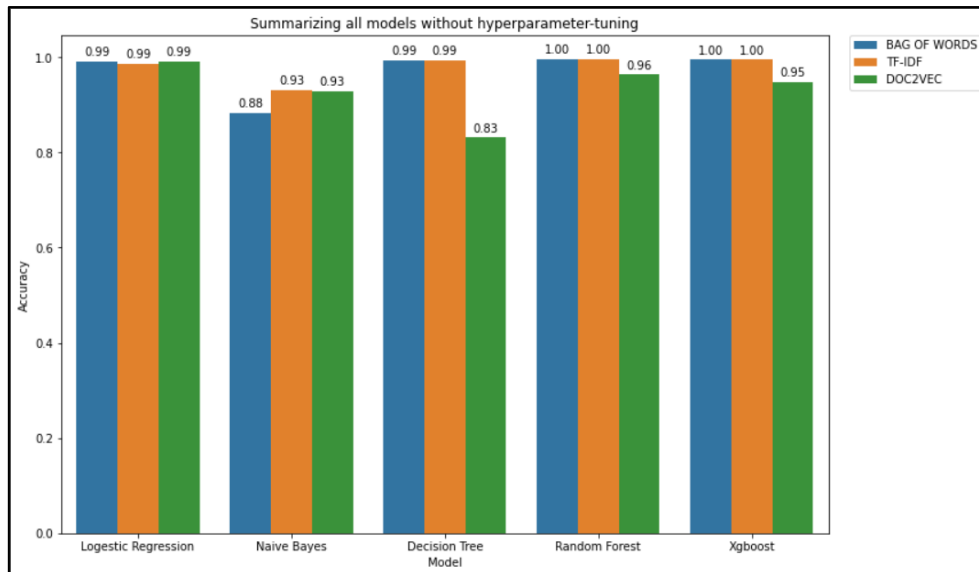


Fig: Graph representation of each model without hyperparameter tuning

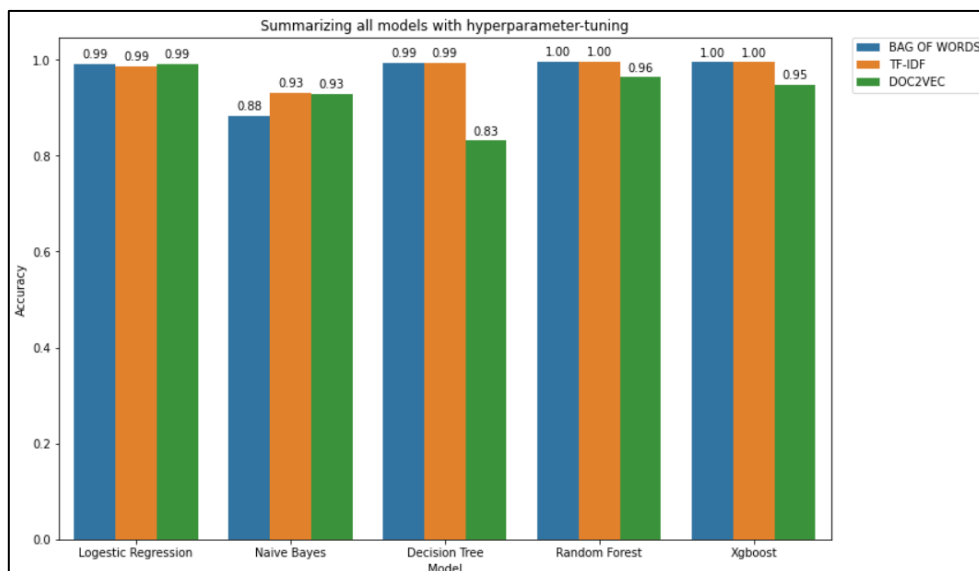


Fig: Graph representation of each model with hyperparameter tuning

As we can see the best result is given by logistic regression and random forest as per accuracy and their log loss. We can also optimize more if given more data.

Comparison logistic regression vs random forest

- Accuracy of logistic regression is 99.26% whereas Random forest accuracy is 99.67%.
- Logistic regression is inherently linear whereas random forest is inherently non-linear.

- Logistic regression is faster(Time complexity : $O(n)$) as compare to random forest(train time - $O(n \cdot \lg(nd))$ and run time – $O(\text{depth})$, where n = number of data points and d as dimension)
- Logistic regression works with probability estimates which are interpretable whereas random forest has lack of interpretation.
- Logistic regression can't handle high dimensional data whereas random forest can easily be handled.
- Logistic regression is less complex as compare to random forest.

4.Error Analysis and Feature Engineering

As we are increasing the number of features in bag of words and tf-idf it is seen a significant accuracy change of about 0.1% to 0.2% as we have taken till 1000 features, we can take up to 5000 to increase somewhat more accuracy but due to computational expensive we are taking features till 1000.

We have also checked for features like word count or characters count per news but that is not giving any better result so removed those features.

As we have seen some of the places doc2vec is not performing well and having much less performance as compare to tf-idf and bag of words, this is because we training our doc2vec on our dataset which is not that much large and due to that it is not able to capture word relationship in the embedding space with limited information.

Here, we have to consider two things which are most important i.e. we need to check whether are model is over-fitting or under-fitting.

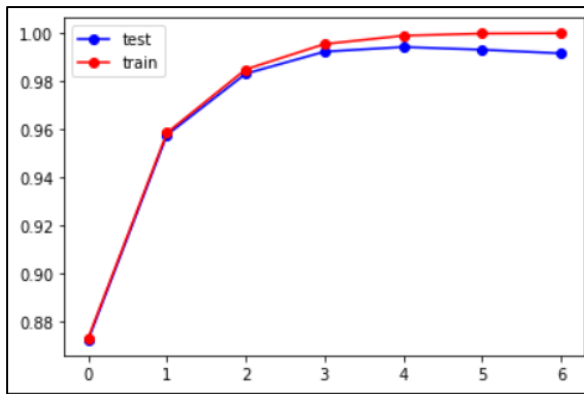
Over-fitting means when we have high training score and low test score and also called as low bias and high variance.

Under-fitting means when we have low training score and low test score and also called as high bias and high variance.

So, we need train score and test score difference should be small to be a good model which requires various engineering techniques and parameter tuning.

4.1 Logistic Regression

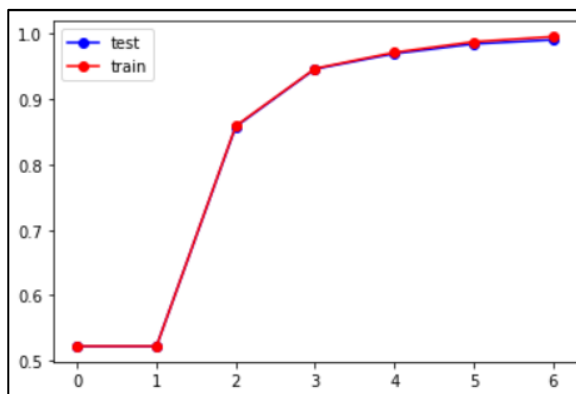
Here, we are doing hyper parameter tuning of parameter 'C' and performed for all the 3 vectorizer – countvectorizer, tf-idf and doc2vec. For hyper parameter tuning we will be using grid search.



Best parameters
gridsearch: {'C': 0.1}

Accuracy: 0.9939420935412027
Error: 0.08953961567126707

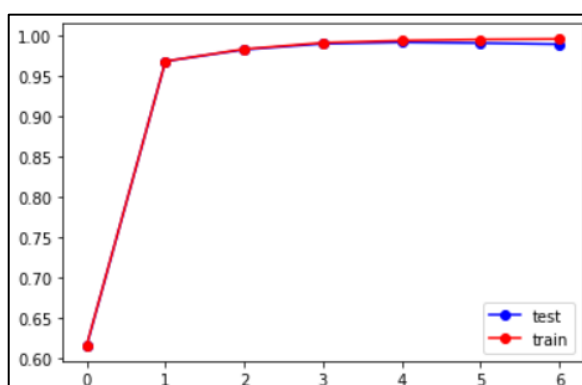
Fig: Countvectorizer



Best parameters
gridsearch: {'C': 10}

Accuracy: 0.9903786191536749
Error: 0.1182632793402163

Fig: tfidfvectorizer



Best parameters
gridsearch: {'C': 0.1}

Accuracy: 0.9914476614699332
Error: 0.10425295392994251

Fig: doc2vec

Key Observation

- In countvectorizer, we got best value for 'C' as 0.1 and we can see from graph also that if we take value more than this then the difference between train

score and test score is increasing which will result in overfitting after certain point.

- In tf-idfvectorizer, we got best value for 'C' as 10 and we can clearly see it is almost overlapping
- In doc2vec, we we got best value for 'C' as 0.1 and here just after 0.1 train score and test score stating deviating.
- So, overall all 3 vectorizer almost giving same accuracy but countvectorizer giving best result across all vectorizer and least error.

4.2 Decision Trees

Here, we are doing hyper parameter tuning of parameters – criterion, max_depth and min_samples_leaf and performed for all the 3 vectorizer – countvectorizer, tf-idf and doc2vec. For hyper parameter tuning we will be using grid search.

```
Best parameters
gridsearch: {'criterion': 'gini', 'max_depth': 7, 'min_samples_leaf': 1}
```

```
Accuracy: 0.9942984409799555
Error: 0.07551077810555669
```

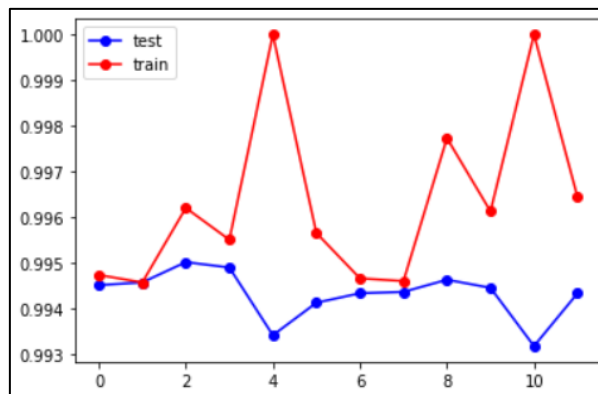


Fig: Countvectorizer

```
Best parameters
gridsearch: {'criterion': 'gini', 'max_depth': 7, 'min_samples_leaf': 7}
```

```
Accuracy: 0.9951893095768374
Error: 0.08064354735456687
```

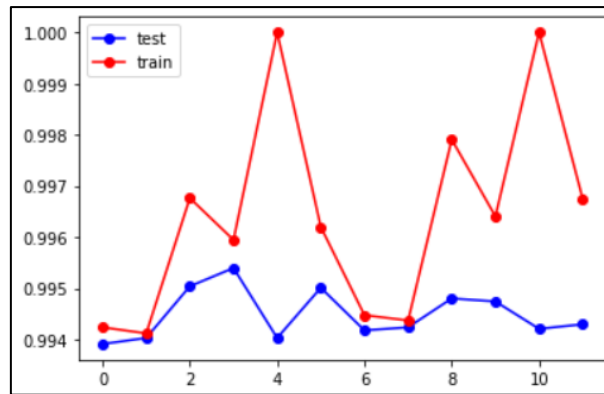


Fig: tfidfvectorizer

```
Best parameters
gridsearch: {'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 1}

Accuracy: 0.8641425389755011
Error: 0.40520063115430865
```

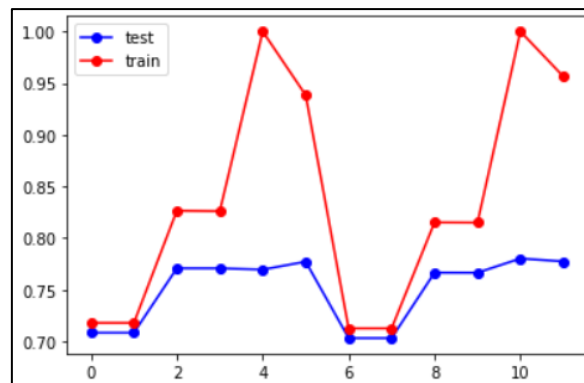


Fig: doc2vec

Key Observation

- In countvectorizer, we got best value for 'criterion' as gini, 'max_depth' as 7, 'min_samples_leaf' as 1 and if we take value more than this then their may be a chance that the difference between train score and test score is increasing which will result in overfitting after certain point. We can see it is almost overlapped at starting.
- In tf-idfvectorizer, we got best value for for 'criterion' as gini, 'max_depth' as 7, 'min_samples_leaf' as 7 and we can clearly see it is overlapping at starting.
- In doc2vec, we we got best value for 'criterion' as entropy, 'max_depth' as None, 'min_samples_leaf' as 1 also train score and test score is deviating and also the accuracy is not increasing much at some point it is overfitting also.
- So, overall the best accuracy is of tfidfvectorizer with least error.

4.3 Random forest

Here, we are doing hyper parameter tuning of parameters – max_depth, min_samples_leaf, min_samples_split and n_estimators and performed for all the 3 vectorizer – countvectorizer, tf-idf and doc2vec. For hyper parameter tuning we will be using grid search.

```
Best parameters
gridsearch: {'max_depth': 25, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 500}

Accuracy: 0.9967037861915368
Error: 0.06401497476408204
```

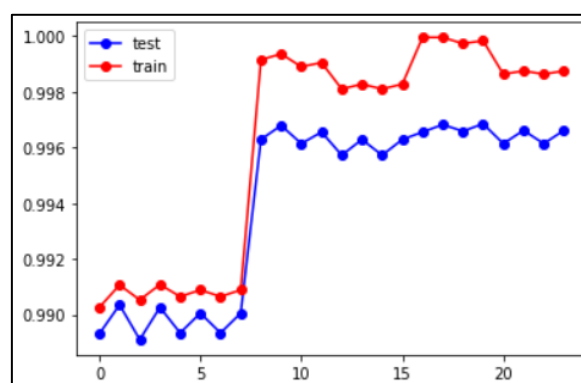


Fig: Countvectorizer

```
Best parameters
gridsearch: {'max_depth': 25, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 500}

Accuracy: 0.9964365256124722
Error: 0.06470675535876773
```

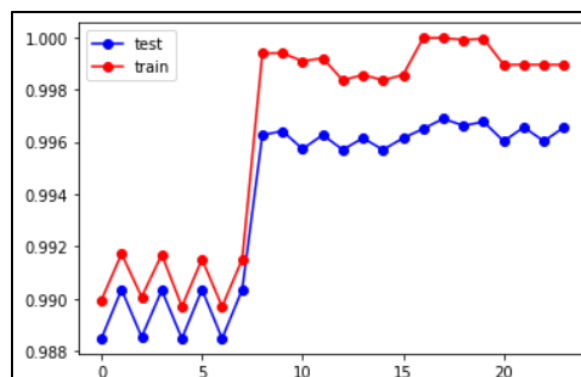


Fig: tfidfvectorizer


```
Best parameters
gridsearch: {'max_depth': 25, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 500}

Accuracy: 0.9716703786191537
Error: 0.18131071015773836
```

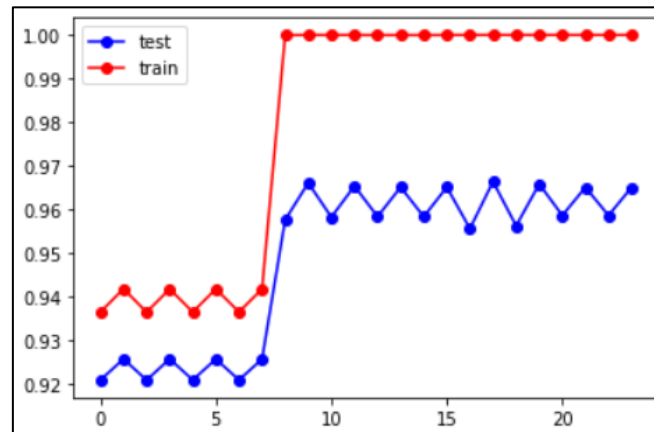


Fig: doc2vec

Key Observation

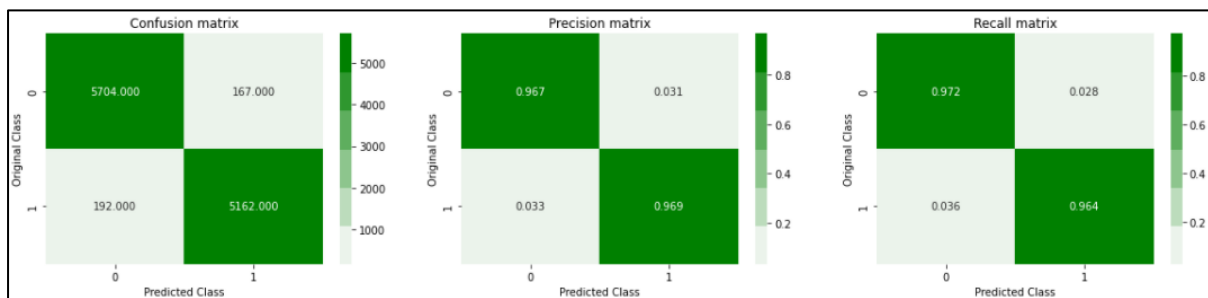
- In countvectorizer, we got best value for max_depth as 25, min_samples_leaf as 1, min_samples_split as 5 and n_estimators as 500, we can clearly see that the difference between train and test score is very less so we don't have a case of overfitting.
- In tf-idfvectorizer, we got best value for for max_depth as 25, min_samples_leaf as 1, min_samples_split as 2 and n_estimators as 500 and we can clearly see it is giving a great result with least error.
- In doc2vec, we we got best value for max_depth as 25, min_samples_leaf as 1, min_samples_split as 2 and n_estimators as 500 also train score and test score don't have much difference, which is better than most the other models.
- So, overall the best accuracy is of countvectorizer with least error.

4.4 BERT

BERT stands for Bidirectional Encoder Representations from Transformers which is a machine learning framework for NLP task. It is pre-trained on Wikipedia and can be fine-tuned later according to dataset need. So, here I have taken pre-trained model 'bert_en_uncased_L-12_H-768_A-12/4' and fine tuned based on my given dataset and got an accuracy of 97%.

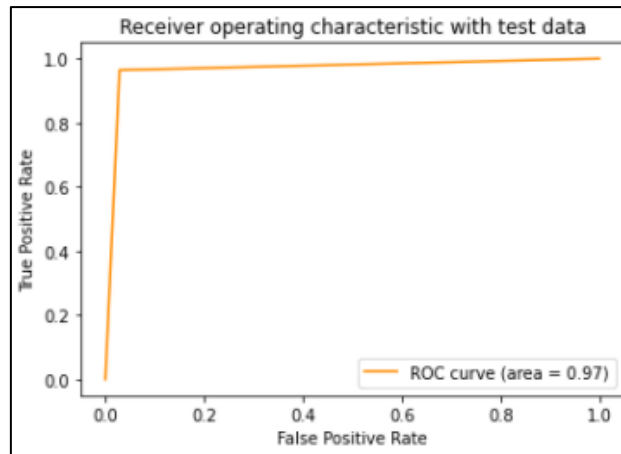
There might be a question why this advanced model does not give accuracy better than classical ML model, this is because we have taken only 128 words from each sentences and 512 words atmost that bert takes input. Although I have not divided sentences into sub text and making a new dataset but that will increase time complexity exponentially but due to computational expensive we are taking bert input as 128 because of that half of the information is diluted. Even though with 128 words from a sentence it gives a very good accuracy

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.97 | 0.97 | 0.97 | 5871 |
| 1 | 0.97 | 0.96 | 0.97 | 5354 |
| accuracy | | | 0.97 | 11225 |
| macro avg | 0.97 | 0.97 | 0.97 | 11225 |
| weighted avg | 0.97 | 0.97 | 0.97 | 11225 |



Key Observation:

- Accuracy is 97% for bert model.
- From confusion matrix we can see 5162 points are correctly classified as true news whereas we have error of 192 points which are misclassified as fake news. Also for 5704 points are correctly classified as fake news but 167 of them are misclassified as true news.
- Taken 128 words as input for each sentence to bert's input. This can be improved but due to computational restriction I have not created sub-text and dividing sentence in 128 sub-text each or 500 sub-text each.



ROC(Receiver operating characteristics) graphs provide a simple way to summarize all of the information. In y-axis we have true positive rate which is same as sensitivity which tells what proportion of true news samples were correctly classified and in x-axis we have false positive rate which is same as (1- specificity) which tells the proportion of fake news samples that were incorrectly classified and are false positive. In this graph showing an area of 0.97 is covered.

5. Deployment and Productionization

5.1 Flask and templates implementation

We have taken model as logistic regression and took pickle file i.e. Logistic_Bow_model.pkl and for vectorizer we choose count vectorizer whose pickle file is CountVectorizer.pkl.

Now we will be using flask framework to showcase our project in a form of website. So, to begin with we will be creating a virtual environment. These virtual environments help us to manage our packages and can have specific version of packages so that in later part of time if there is any version update this will not affect our project working. As here I am using a anaconda prompt so, command line will be based on that :

```
Step:1 Creating virtual environment
      conda create --name myenv
Step:2 Activate your environment
      conda activate myenv
Step:3 Open your project using visual studio (checked the path)
      code .
```

First we will be creating python file app.py and open a terminal and activate my environment using activate myenv then whatever will the packages we can install. First to install pip we will be using conda install pip after that we can install any

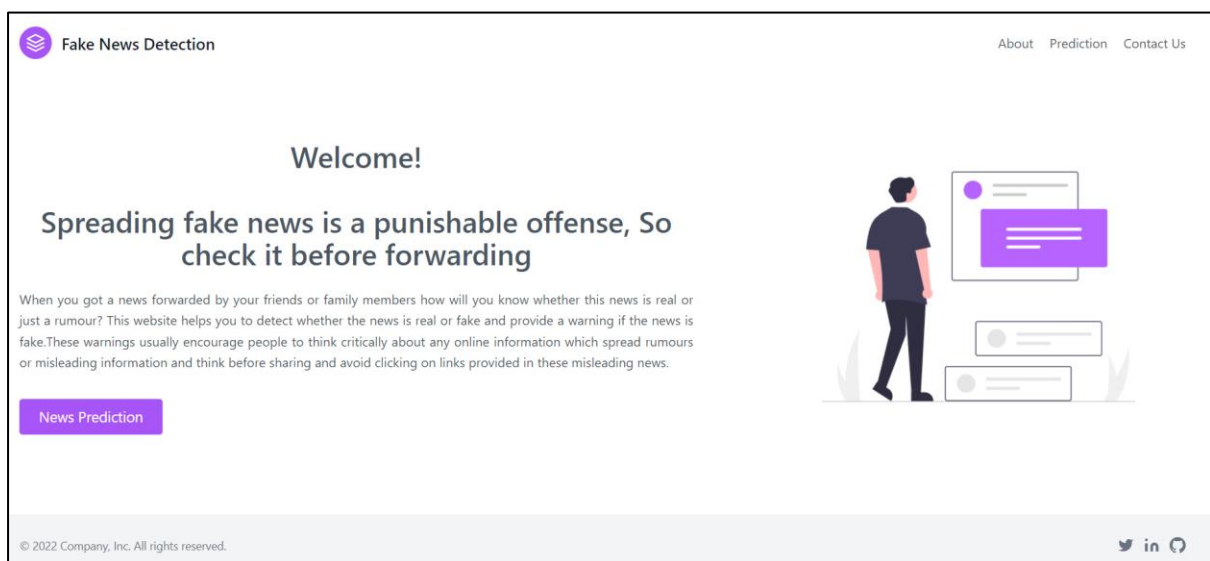
packages in this environment. We will be installing these by `pip install package-name`. Below given the list of packages name:

```
bz2file==0.98
Flask==2.0.3
gunicorn==20.1.0
Jinja2==3.0.3
nlTK==3.7
pandas==1.4.1
sklearn==0.0
```

First, we will be creating our html pages for that I have used tailwind css which packed with classes as we can give classes to our html tags and it will understand which css to implement and after implementation we can get our css file using command line. So, I will be creating 5 html pages which are:

- index.html (this will be our front page)
- prediction.html (where we will going to do prediction of news)
- About.html (about section of website)
- Contact.html (contact section and feedback section)
- Feedback.html (when we submit feedback from contact page it will redirect to this feedback page)

Below is shown our index.html page which has 3 sections i.e. Header section, Body section and footer section. In header section we have navigation bar which help in redirect to a certain page which will be defined in app.py. In body section whatever we see in the middle region and have button to redirect to prediction page.



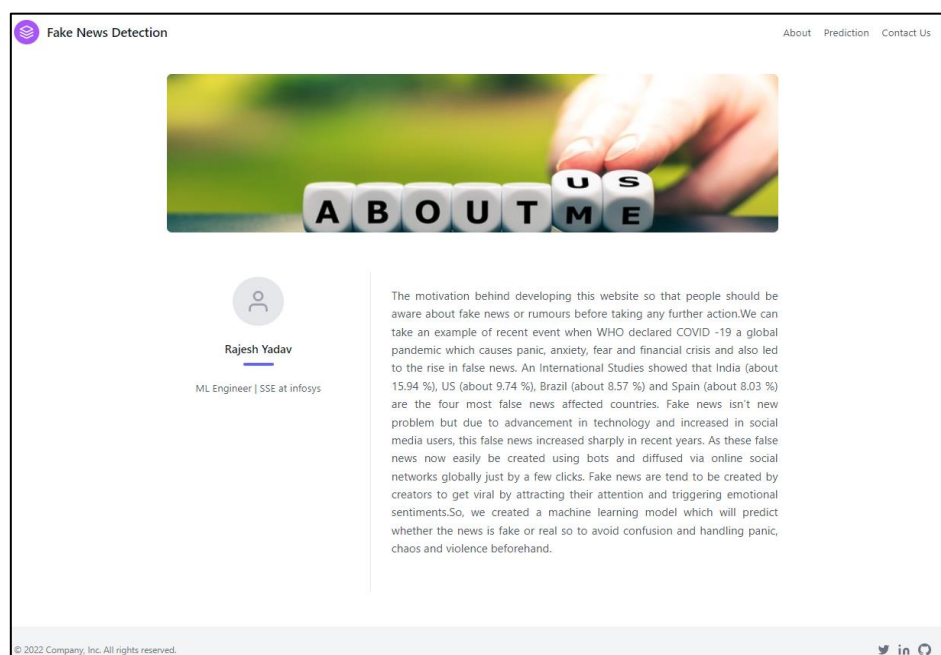
Below code is in app.py which help to redirect to the index.html page and also we set it as a home page.

```
@app.route("/")
def Home():
    return render_template("index.html")
```

Then we have About.html page for which in app.py we defined a get request method as it will pull up about page and display over.

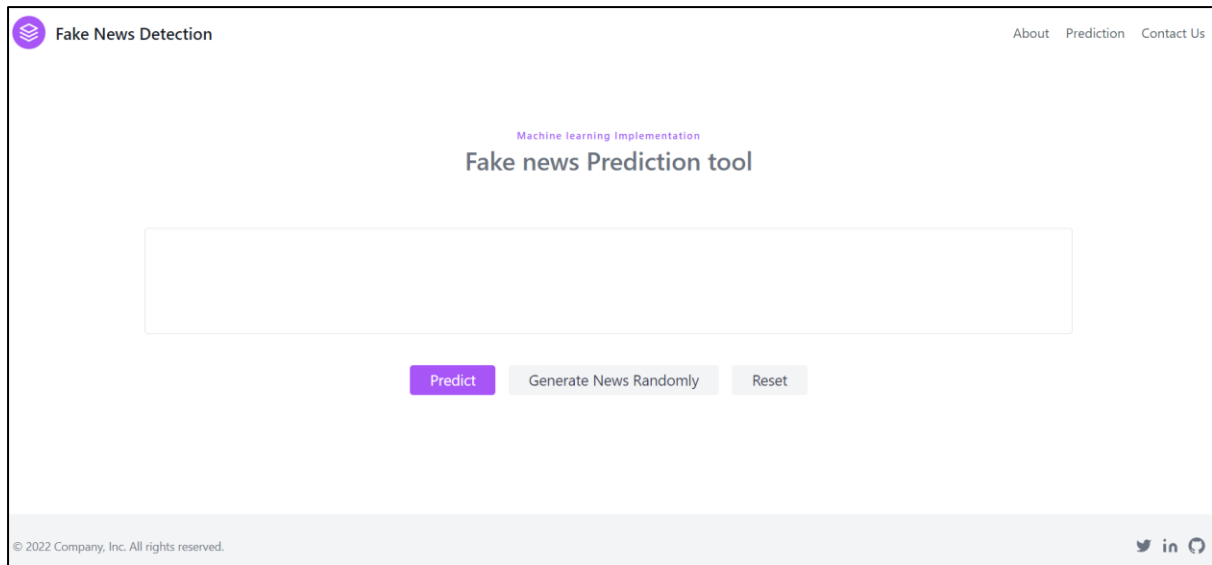
```
@app.route("/about", methods=['GET'])
def about():
    return render_template("About.html")
```

Header section and footer section is same for all of them only body section will be different.



Now comes the important part of the project where we will do the prediction of news whether it is fake or real.

In this we have 3 buttons so Predict button will give result as fake or real news, generate news randomly will take random news from test dataset or unseen dataset and reset button will reset the page so that we can start with new values.



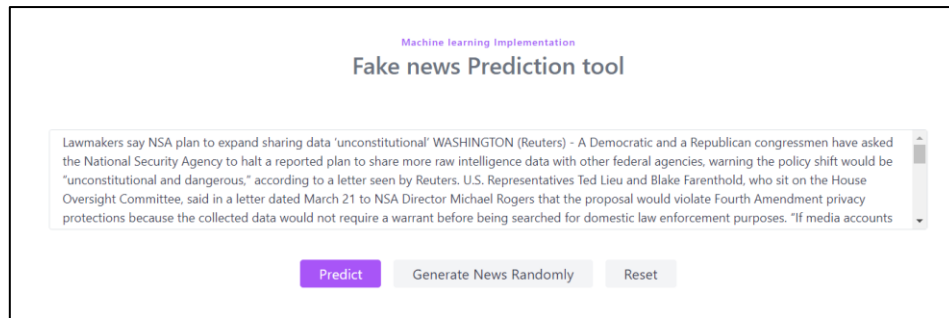
In code, news_submit_a is a name or id for Predict button, same for generate news randomly button has news_submit_b and reset button has news_submit_c.

These entire buttons will be in form tag where we will have action and method defined.

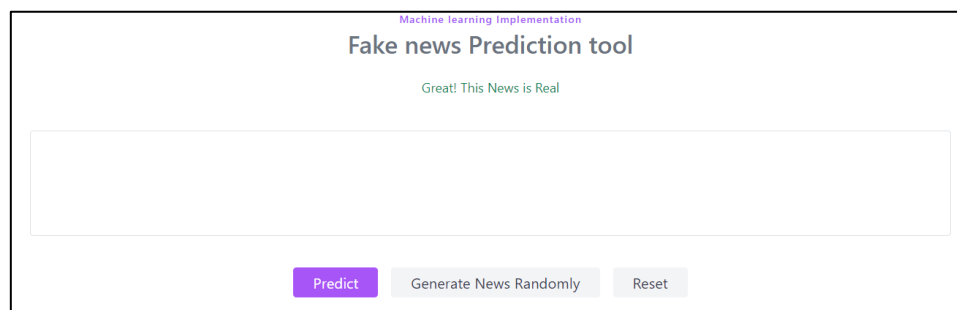
```
<form action="/predict" method="POST">
```

```
@app.route("/predict", methods=['GET', 'POST'])
def Newsprediction():
    if request.method == "POST":
        if request.form.get("news_submit_a"):
            retrieved_news = str(request.form['news_details'])
            print(retrieved_news)
            text_preprocessed = text_preprocessing(retrieved_news)
            predict = logistic_model.predict(text_vectorizer.transform([text_preprocessed]))[0]
            print(predict)
            return render_template("prediction.html", predicted_news = predict)
        elif request.form.get("news_submit_b"):
            data = pd.read_csv("test_data.csv.gz", compression='gzip')
            index = randrange(0, len(data)-1, 1)
            news = data.loc[index].title + " " + data.loc[index].text
            print(data.loc[index].id)
            return render_template("prediction.html", predict_news = news)
        elif request.form.get("news_submit_c"):
            return render_template("prediction.html")
    else:
        return render_template("prediction.html")
```

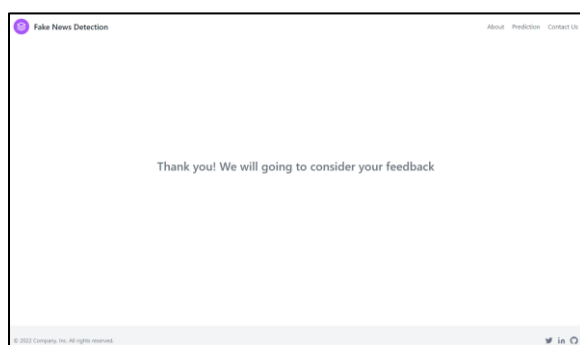
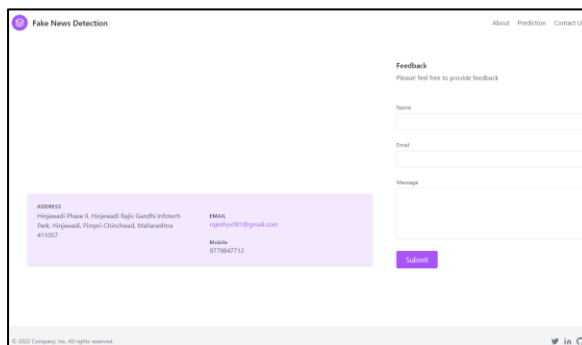
So, we can input manually or generate randomly, when we click on generate news randomly post method will be triggered within form in prediction.html then it will go to app.py and get into elif loop where it has news_submit_b then we get the data from test_data which was in compressed state then using randrange we will be getting random id and with that id we get our news and get rendered as shown below.



Now, after we have our news we will press the predict button which will trigger to app.py and get into if loop where it has news_submit_a then it retrieve data from text box whose id in html page is news_details after that we will clean or text_preprocess our data and do the countvectorization and predict using logistic model and render a result to the predict page which result in Real news.



This one is Contact.html and feedback.html. In contact.html we enter a feedback and after submitting redirect to the feedback.html with a message.

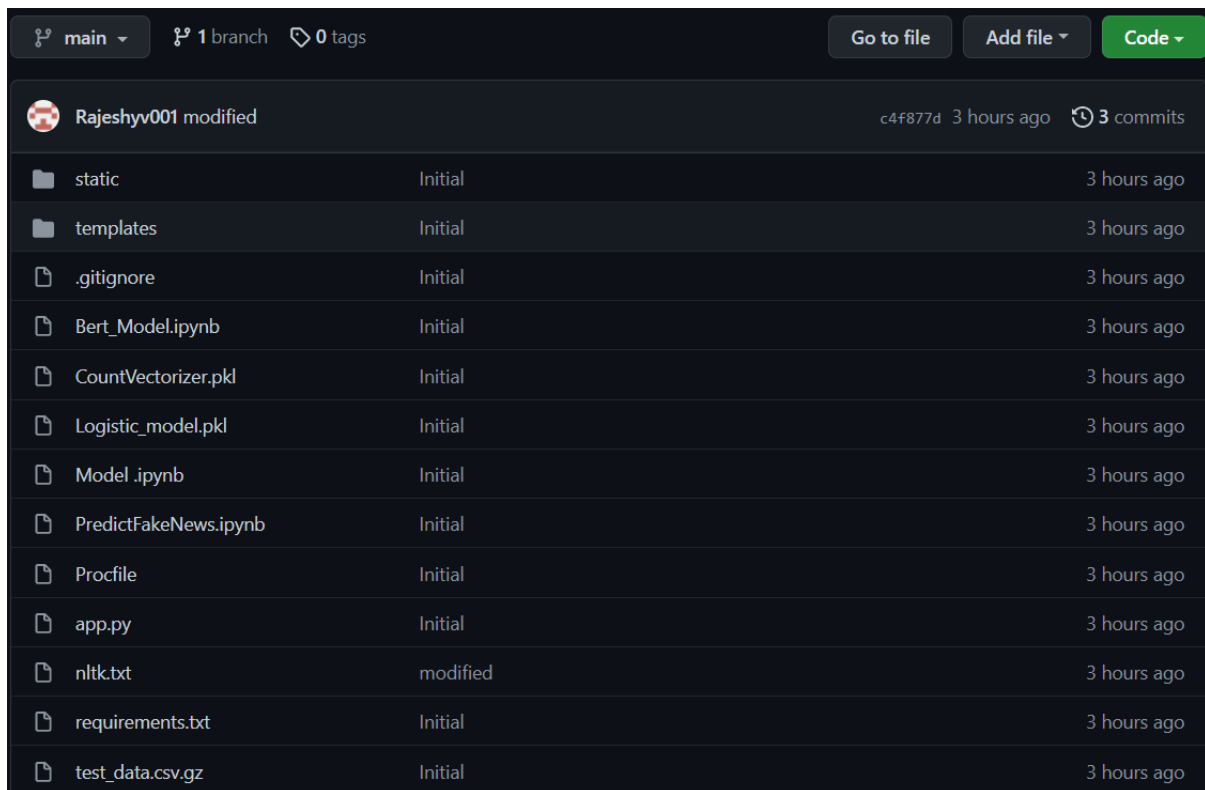


5.2 Uploading project to Github

Go to the project directory and here we will be using git bash command line.

- **git init** (initializing your git)
- **git status** (check the status of files which are tracked or not)
- **git add .** (adding all the files at once)
- **git commit -m 'initial commit'** (provide commit to each file)
- **git remote add origin** <https://github.com/Rajeshyv001/Fakenews detection.git> (connecting with the github repository)

- **git branch –M main** (moving to main branch)
- **git push –u origin main** (uploading all the files to main branch)



| main 1 branch 0 tags | | | Go to file Add file Code |
|-----------------------|----------|---------------------|--------------------------|
| Rajeshyv001 modified | | c4f877d 3 hours ago | 3 commits |
| static | Initial | 3 hours ago | |
| templates | Initial | 3 hours ago | |
| .gitignore | Initial | 3 hours ago | |
| Bert_Model.ipynb | Initial | 3 hours ago | |
| CountVectorizer.pkl | Initial | 3 hours ago | |
| Logistic_model.pkl | Initial | 3 hours ago | |
| Model.ipynb | Initial | 3 hours ago | |
| PredictFakeNews.ipynb | Initial | 3 hours ago | |
| Procfile | Initial | 3 hours ago | |
| app.py | Initial | 3 hours ago | |
| nltk.txt | modified | 3 hours ago | |
| requirements.txt | Initial | 3 hours ago | |
| test_data.csv.gz | Initial | 3 hours ago | |

5.3 Deploying project to Heroku

I have selected heroku to deploy my project as it simple to use and have ready to use environment which allows our code to deploy faster and the best thing about it we just need to follow 3 simple steps and good to go.

Step 1: login to heroku and create a app.
 Step 2: connect your github
 Step 3: deploy you project

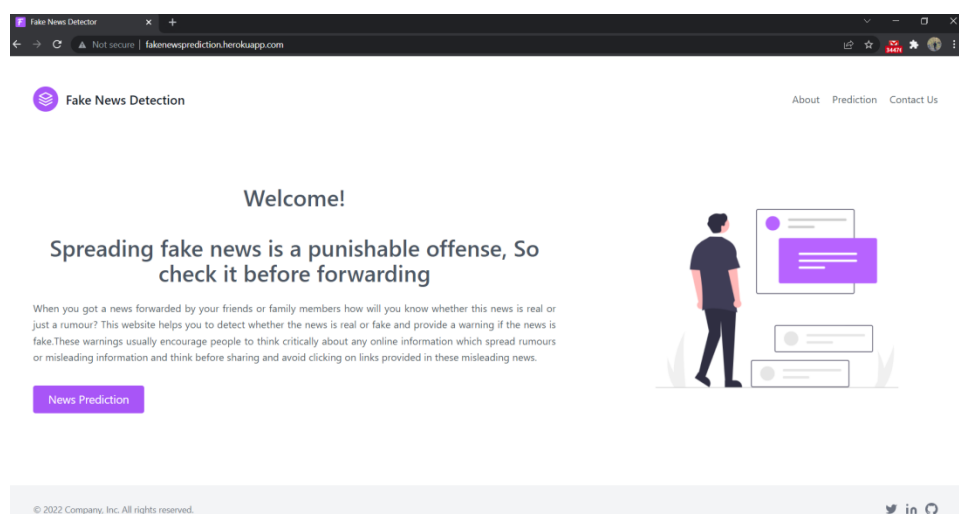
We will need this requirement.txt file which makes heroku understand what libraries needed to install.

```
Flask==2.0.3
gunicorn==20.1.0
Jinja2==3.0.3
nltk==3.7
pandas==1.4.1
sklearn==0.0
```


Also for NLP based project we have to create a file for nltk as nltk.txt which will guide heroku which nltk files needed from corpora to work with text data. So, these libraries will get install in heroku:

```
punkt  
stopwords  
wordnet  
omw-1.4
```

Here, in url (<https://fakenewsprediction.herokuapp.com>) we can see that our project is deployed and is live.



5.4 Problem faced while deploying

The main problem I had faced was working with countvectorizer when created pickle file that was around 250 MB.

First Attempt : Failed

First problem was to upload that big pickle file to github for that case I used Git LFS (Large file storage) which is used to store large files after that when tried to deploy that to heroku I was getting an application error. So, I looked in a log file getting a R14 memory leakage error means our project was using more memory what is not provided by heroku on free subscription.

Second Attempt : Failed

This time I stored my countvectorizer to amazon S3 bucket which is used to store a large file and can be accessed from anywhere by simply providing access key and secret key which is created through IAM (AWS Identity and Access Management) and then using a library boto3 to retrieve the data. In below code first we connect to

the server then we will get an object where our file saved by giving bucket as bucketname and key as filename. So, from that we will be retrieving Body where all the information is present and load it through cPickle.

```
s3client = boto3.client('s3', region_name='us-east-1',aws_access_key_id=ACCESS_KEY, aws_secret_access_key=SECRET_KEY)
def get_vectorizer(s3client):
    response1 = s3client.get_object(Bucket='rajeshfakenewsdata', Key='CountVectorizer.pkl')
    body_string = response1['Body'].read()
    text_vectorizer = cPickle.loads(body_string)
    return text_vectorizer
```

Then this time I tried to deploy on AWS but still their also I was getting an option as killed as my project taking more ram for execution.

Third Attempt : Success

Now, I get back to my countvectorizer start making changes to it so that its size will reduce and will take less ram to work. This was my initial countvectorizer.

```
text_vectorizer = CountVectorizer(ngram_range=(1,3), min_df=10, max_features=1000)
```

The changes I made just removing ngram and min_df and taking more features.

```
text_vectorizer = CountVectorizer(max_features=3000)
```

Even by this change our accuracy increased by 0.08% with better precision. For this when created a pickle file it was around 3.5 MB. By this I was able to deploy this app on heroku without taking much ram usage.

5.5 Limitations

- This system not tend to work very well in real world aspect as here we don't have variety of data and count vectorizer is not designed to take much semantic meaning from the given words.
- As we have bert model it is giving good amount of work although we need to provide more number of words by dividing sentences into sub sentences and providing respective labels this way we create our news data but problem with bert model it takes lot of time to train.
- So, the solution is we should have variety of data from news section and need to do word embedding using word2vec which will form a semantic meaning so best thing we can use LSTM to work with these data.

References:

- <https://www.theinnovationmode.com/the-innovation-blog/misinformation-online-a-solution-powered-by-state-of-the-art-tech>
- <https://www.firstpost.com/india/how-heavy-internet-usage-and-poor-digital-literacy-made-india-worlds-top-source-of-misinformation-on-covid-19-9966471.html>
- <https://www.brookings.edu/research/how-to-combat-fake-news-and-disinformation/>
- <https://towardsdatascience.com/detecting-fake-news-with-and-without-code-dd330ed449d9>
- <https://towardsdatascience.com/performance-metrics-confusion-matrix-precision-recall-and-f1-score-a8fe076a2262>
- <https://www.hindawi.com/journals/complexity/2020/8885861/>
- <https://towardsdatascience.com/all-machine-learning-models-explained-in-6-minutes-9fe30ff6776a>