DEEP LEARNING

Fake News Classifier

Introduction

News media have long been a vital channel for disseminating information about world events to the public. People often trust that the news they consume is accurate and truthful. However, there have been instances where even reputable news outlets have had to retract stories that were later found to be false. The implications of false news can be profound, affecting not only individuals and governments but also entire economies. A single news story can sway public sentiment and influence political and economic landscapes significantly. This underscores the importance of distinguishing fake news from genuine news.

Natural Language Processing (NLP) tools have emerged as powerful solutions to this problem. By leveraging historical data, NLP can help identify and filter out fake news, ensuring that the information reaching the public is credible. With the advent of these technologies, the integrity of news dissemination is increasingly safeguarded, contributing to a more informed and stable society.

Problem Statement

The authenticity of information is a critical issue that impacts businesses and society at large, whether in printed or digital form. The rapid spread of information on social networks can amplify the effects of inaccurate or false news, leading to significant real-world consequences within minutes for millions of users. This problem has garnered public attention and has prompted efforts to develop solutions to mitigate the spread of fake news.

Historically, sensationalism—through eye-catching and often misleading headlines—has been used to capture audience attention and drive information sales. This practice has persisted across all forms of media. However, the advent of social networking websites has significantly amplified the reach and speed at which information spreads. Inaccurate or distorted information now has the potential to cause substantial impact almost instantaneously.

This amplification effect poses a substantial challenge: ensuring the accuracy and credibility of information in an age where news can go viral within moments. Addressing this challenge is essential to prevent the societal, political, and economic disruptions that can result from the spread of fake news. Natural Language Processing offers promising approaches to tackle this issue by analyzing and verifying news content, thereby maintaining the integrity of information dissemination in today's fast-paced digital world.

Objective

- Accurately classify news articles as either fake or true.
- Perform extensive Exploratory Data Analysis (EDA) to understand the dataset.
- Select and build a robust model for news classification.
- Leverage advanced Natural Language Processing (NLP) techniques.
- Ensure high accuracy and reliability in distinguishing fake news from real news.

Import Libraries

Let's import all necessary libraries for the analysis and along with it let's bring down our dataset

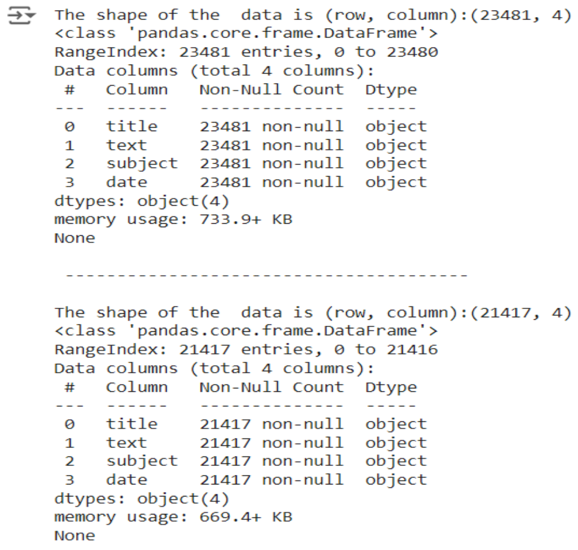
```
#Basic libraries
import pandas as pd
```

```python
import numpy as np

#Visualization libraries
import matplotlib.pyplot as plt
from matplotlib import rcParams
import seaborn as sns
from textblob import TextBlob
from plotly import tools
import plotly.graph_objs as go
from plotly.offline import iplot
%matplotlib inline
plt.rcParams['figure.figsize'] = [10, 5]
import cufflinks as cf
cf.go_offline()
cf.set_config_file(offline=False, world_readable=True)

#NLTK libraries
import nltk
import re
import string
from nltk.corpus import stopwords
from wordcloud import WordCloud,STOPWORDS
from nltk.stem.porter import PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
# Machine Learning libraries
import sklearn
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
#Metrics libraries
from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
#Miscellanous libraries
from collections import Counter

#Ignore warnings
import warnings
warnings.filterwarnings('ignore')
#Deep learning libraries
from tensorflow.keras.layers import Embedding, Bidirectional, Dense, LSTM, Dropout, BatchNormalization
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.text import one_hot
from keras.callbacks import EarlyStopping, LearningRateScheduler
import tensorflow as tf
from keras import Sequential
```

Importing the dataset

Let's welcome our dataset and see what's inside the box

```
#reading the fake and true datasets
fake_news = pd.read_csv('Fake.csv')
true_news = pd.read_csv('True.csv')
# print shape of fake dataset with rows and columns and information
print ("The shape of the  data is (row, column):"+ str(fake_news.shape))
print (fake_news.info())
print("\n -------------------------------------- \n")
# print shape of true dataset with rows and columns and information
print ("The shape of the  data is (row, column):"+ str(true_news.shape))
print (true_news.info())
```



```
The shape of the  data is (row, column):(23481, 4)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23481 entries, 0 to 23480
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   title    23481 non-null  object
 1   text     23481 non-null  object
 2   subject  23481 non-null  object
 3   date     23481 non-null  object
dtypes: object(4)
memory usage: 733.9+ KB
None

 --------------------------------------

The shape of the  data is (row, column):(21417, 4)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21417 entries, 0 to 21416
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   title    21417 non-null  object
 1   text     21417 non-null  object
 2   subject  21417 non-null  object
 3   date     21417 non-null  object
dtypes: object(4)
memory usage: 669.4+ KB
None
```

Dataset Details

This metadata has two csv files where one dataset contains fake news and the other contains true/real news and has nearly **23481 fake news and 21417 true news**

Description of columns in the file:

- title- contains news headlines
- text-contains news content/article
- subject- type of news
- date- date the news was published

Preprocessing and Cleaning

We have to perform certain preprocessing steps before performing EDA and giving the data to the model.Let's begin with creating the output column

Creating the target column:

Let's create the target column for both fake and true news. Here we are gonna denote the target value as '0' incase of fake news and '1' incase of true news

```
#Target variable for fake news
fake_news['output']=0
```

```
#Target variable for true news
true_news['output']=1
```

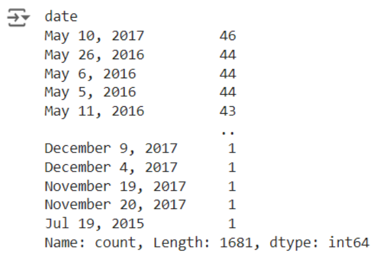
Concatenating title and text of news:

```
#Concatenating and dropping for fake news
fake_news['news']=fake_news['title']+fake_news['text']
fake_news=fake_news.drop(['title', 'text'], axis=1)
#Concatenating and dropping for true news
true_news['news']=true_news['title']+true_news['text']
true_news=true_news.drop(['title', 'text'], axis=1)
#Rearranging the columns
fake_news = fake_news[['subject', 'date', 'news','output']]
true_news = true_news[['subject', 'date', 'news','output']]
```

Converting the date columns to datetime format:

We can use pd.datetime to convert our date columns to date format we desire. But there was a problem,especially in fake_news date column. Let's check the value_counts() to see what lies inside.

```
fake_news['date'].value_counts()
```



```
date
May 10, 2017          46
May 26, 2016          44
May 6, 2016           44
May 5, 2016           44
May 11, 2016          43
                      ..
December 9, 2017       1
December 4, 2017       1
November 19, 2017      1
November 20, 2017      1
Jul 19, 2015           1
Name: count, Length: 1681, dtype: int64
```

we had links and news headline inside the date column which can give us trouble when converting to datetime format. So let's remove those records from the column.

```
#Removing links and the headline from the date column
fake_news=fake_news[~fake_news.date.str.contains("http")]
fake_news=fake_news[~fake_news.date.str.contains("HOST")]
```

Only fake news dataset had an issue with date column,Now let's proceed with converting the date column to datetime format

```
#Converting the date to datetime format
fake_news['date'] = pd.to_datetime(fake_news['date'], format='mixed', errors='coerce')
# Use format='mixed' to handle multiple formats and errors='coerce' to handle invalid dates
true_news['date'] = pd.to_datetime(true_news['date'], format='mixed', errors='coerce')
```

Appending two datasets:

When we are providing a dataset for the model, we have to provide it as a single file. So it's better to append both true and fake news data and preprocess it further and perform EDA.

```
frames = [fake_news, true_news]
news_dataset = pd.concat(frames)
news_dataset
```

	subject	date	news	output
0	News	2017-12-31	Donald Trump Sends Out Embarrassing New Year'...	0
1	News	2017-12-31	Drunk Bragging Trump Staffer Started Russian ...	0
2	News	2017-12-30	Sheriff David Clarke Becomes An Internet Joke...	0
3	News	2017-12-29	Trump Is So Obsessed He Even Has Obama's Name...	0
4	News	2017-12-25	Pope Francis Just Called Out Donald Trump Dur...	0
...	...	...	...	...
21412	worldnews	2017-08-22	'Fully committed' NATO backs new U.S. approach...	1
21413	worldnews	2017-08-22	LexisNexis withdrew two products from Chinese ...	1
21414	worldnews	2017-08-22	Minsk cultural hub becomes haven from authorit...	1
21415	worldnews	2017-08-22	Vatican upbeat on possibility of Pope Francis ...	1
21416	worldnews	2017-08-22	Indonesia to buy $1.14 billion worth of Russia...	1

44888 rows × 4 columns

Text Processing

This is an important phase for any text analysis application.There will be many unuseful content in the news which can be an obstacle when feeding to a machine learning model.Unless we remove them the machine learning model doesn't work efficiently. Lets go step by step.

News-Punctuation Cleaning

Let's begin our text processing by removing the punctuations

```
#Creating a copy
clean_news=news_dataset.copy()
def review_cleaning(text):
    '''Make text lowercase, remove text in square brackets,remove links,remove punctuation
    and remove words containing numbers.'''
    text = str(text).lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text = re.sub('<.*?>+', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub('\n', '', text)
    text = re.sub('\w*\d\w*', '', text)
    return text
clean_news['news']=clean_news['news'].apply(lambda x:review_cleaning(x))
clean_news.head()
```

	subject	date	news	output
0	News	2017-12-31	donald trump sends out embarrassing new year'...	0
1	News	2017-12-31	drunk bragging trump staffer started russian ...	0
2	News	2017-12-30	sheriff david clarke becomes an internet joke...	0
3	News	2017-12-29	trump is so obsessed he even has obama's name...	0
4	News	2017-12-25	pope francis just called out donald trump dur...	0

We have removed all punctuation in our news column

News-Stop words:

A stop word is a commonly used word (such as "the", "a", "an", "in") that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query. We would not want these words to take up space in our database, or taking up valuable processing time. For this, we can remove them easily, by storing a list of words that you consider to stop words. NLTK(Natural Language Toolkit) in python has a list of stopwords stored in 16 different languages.

For our project, we are considering the english stop words and removing those words

```
import nltk
```

```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

```
stop = stopwords.words('english')
clean_news['news'] = clean_news['news'].apply(lambda x: ' '.join([word for word in x.split() if word not in
(stop)]))
clean_news.head()
```

	subject	date	news	output
0	News	2017-12-31	donald trump sends embarrassing new year's eve...	0
1	News	2017-12-31	drunk bragging trump staffer started russian c...	0
2	News	2017-12-30	sheriff david clarke becomes internet joke thr...	0
3	News	2017-12-29	trump obsessed even obama's name coded website...	0
4	News	2017-12-25	pope francis called donald trump christmas spe...	0

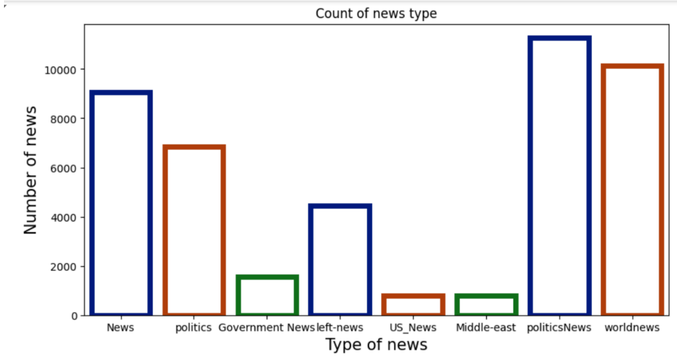
We have removed all the stop words in the review column

Story Generation and Visualization from news

we will complete do exploratory data analysis on news such as ngram analysis and understand which are all the words,context which are most likely found in fake news

Count of news subject:

```
#Plotting the frequency plot
ax = sns.countplot(x="subject", data=clean_news,
              facecolor=(0, 0, 0, 0),
              linewidth=5,
                edgecolor=sns.color_palette("dark", 3))
#Setting labels and font size
ax.set(xlabel='Type of news', ylabel='Number of news',title='Count of news type')
ax.xaxis.get_label().set_fontsize(15)
ax.yaxis.get_label().set_fontsize(15)
```

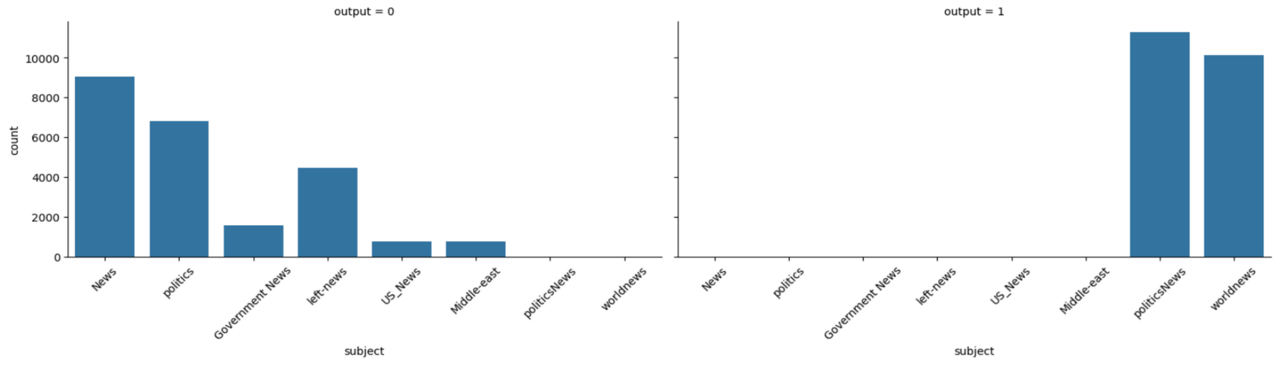


Count of news subject based on true or fake:

```
g = sns.catplot(x="subject", col="output",
                data=clean_news, kind="count",
                height=4, aspect=2)
```

```
#Rotating the xlabels
g.set_xticklabels(rotation=45)
```

```
<seaborn.axisgrid.FacetGrid at 0x7c8ebc986e90>
```

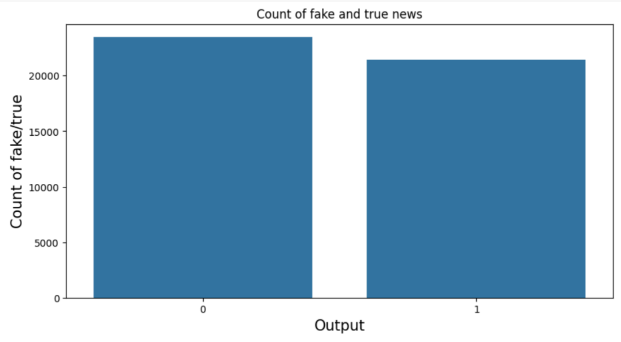


Insights:

- Fake news are all over the category except politics and world news
- True news are present only in politics and world news and the count is high
- THIS IS A HIGHLY BIASED DATASET and we can expect higher accuracy which doesn't signify it is a good model considering the poor quality of dataset

Count of fake news and true news:

```
ax=sns.countplot(x="output", data=clean_news)
#Setting labels and font size
ax.set(xlabel='Output', ylabel='Count of fake/true',title='Count of fake and true news')
ax.xaxis.get_label().set_fontsize(15)
ax.yaxis.get_label().set_fontsize(15)
```



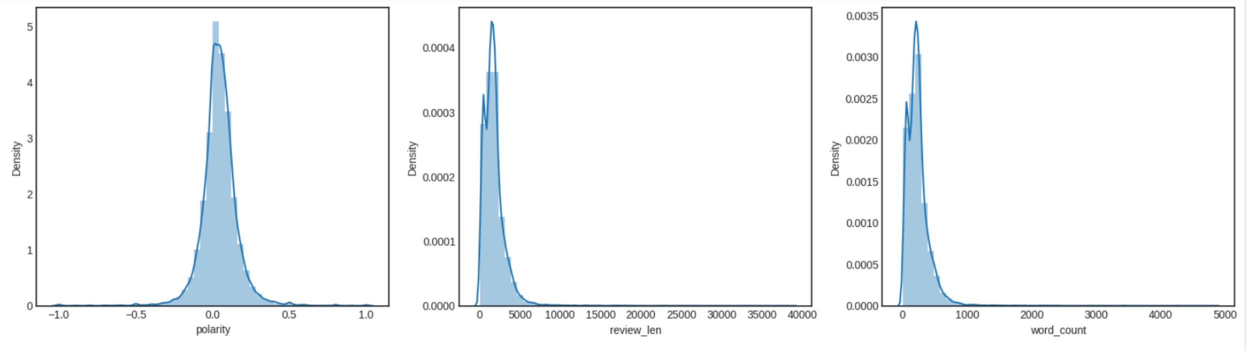
Insights:

- We have a pretty much balanced data
- But the count of fake news is higher than the true news but not on a greater extent

Deriving new features from the news

1. Polarity: The measure which signifies the sentiment of the news
2. Review length: Length of the news(number of letters and spaces)
3. Word Count: Number of words in the news

```
#Extracting the features from the news
clean_news['polarity'] = clean_news['news'].map(lambda text: TextBlob(text).sentiment.polarity)
clean_news['review_len'] = clean_news['news'].astype(str).apply(len)
clean_news['word_count'] = clean_news['news'].apply(lambda x: len(str(x).split()))
#Plotting the distribution of the extracted feature
plt.figure(figsize = (20, 5))
plt.style.use('seaborn-white')
plt.subplot(131)
sns.distplot(clean_news['polarity'])
fig = plt.gcf()
plt.subplot(132)
sns.distplot(clean_news['review_len'])
fig = plt.gcf()
plt.subplot(133)
sns.distplot(clean_news['word_count'])
fig = plt.gcf()
```



Insights:

- Most of the polarity are neutral, neither it shows some bad news nor much happy news
- The word count is between 0-1000 and the length of the news are between 0-5000 and few near 10000 words which could be an article

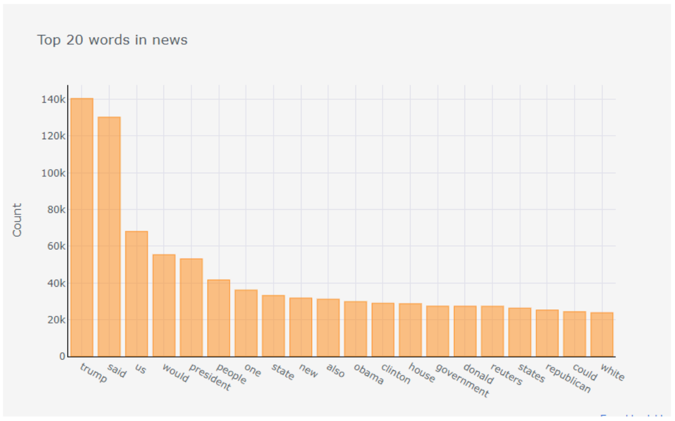
N-gram analysis

Top 20 words in News:

Let's look at the top 20 words from the news which could give us a brief idea on what news are popular in our dataset

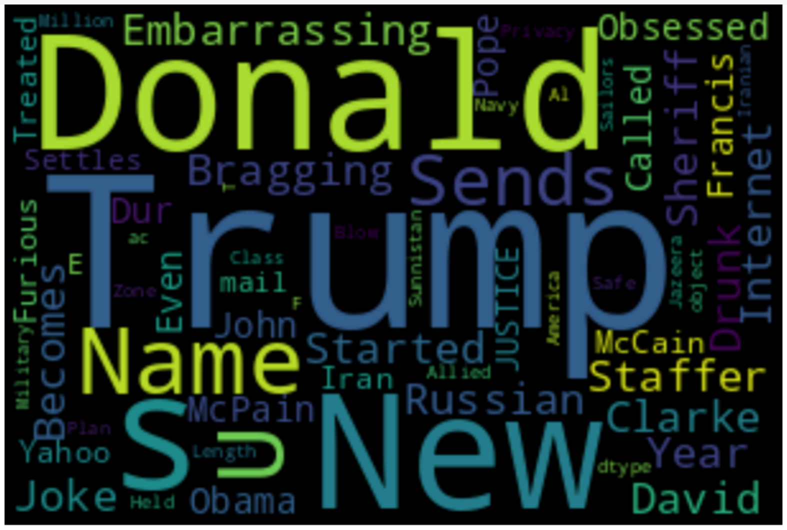
```
#Function to get top n words
def get_top_n_words(corpus, n=None):
    vec = CountVectorizer().fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]
#Calling function and return only top 20 words
common_words = get_top_n_words(clean_news['news'], 20)
#Printing the word and frequency
for word, freq in common_words:
    print(word, freq)
#Creating the dataframe of word and frequency
df1 = pd.DataFrame(common_words, columns = ['news' , 'count'])
#Group by words and plot the sum
df1.groupby('news').sum()['count'].sort_values(ascending=False).iplot(
    kind='bar', yTitle='Count', linecolor='black', title='Top 20 words in news')
```

```
trump 140400
said 130258
us 68081
would 55422
president 53189
people 41718
one 36146
state 33190
new 31799
also 31209
obama 29881
clinton 29003
house 28716
government 27392
donald 27376
reuters 27348
states 26331
republican 25287
could 24356
white 23823
```



WordCloud of Fake and True News:

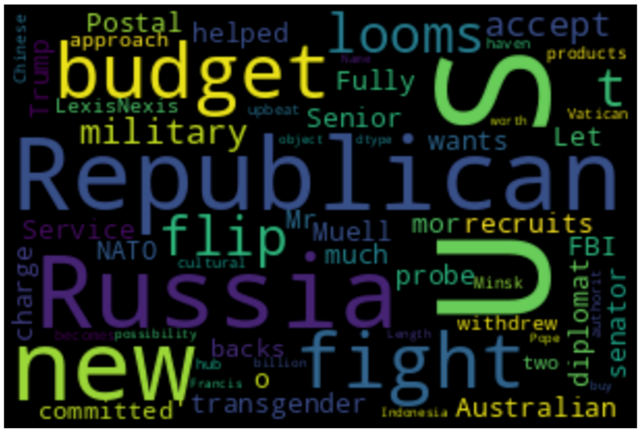
```
text = fake_news["news"]
wordcloud = WordCloud(
    width = 300,
    height = 200,
    background_color = 'black',
    stopwords = STOPWORDS).generate(str(text))
fig = plt.figure(
    figsize = (14, 8),
    facecolor = 'k',
    edgecolor = 'k')
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```



Insights:

- Most of the fake news revolves around Donald Trump and America
- There are also fake news about privacy, internet etc.,

```
text = true_news["news"]
wordcloud = WordCloud(
    width = 300,
    height = 200,
    background_color = 'black',
    stopwords = STOPWORDS).generate(str(text))
fig = plt.figure(
    figsize = (14, 8),
    facecolor = 'k',
    edgecolor = 'k')
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```

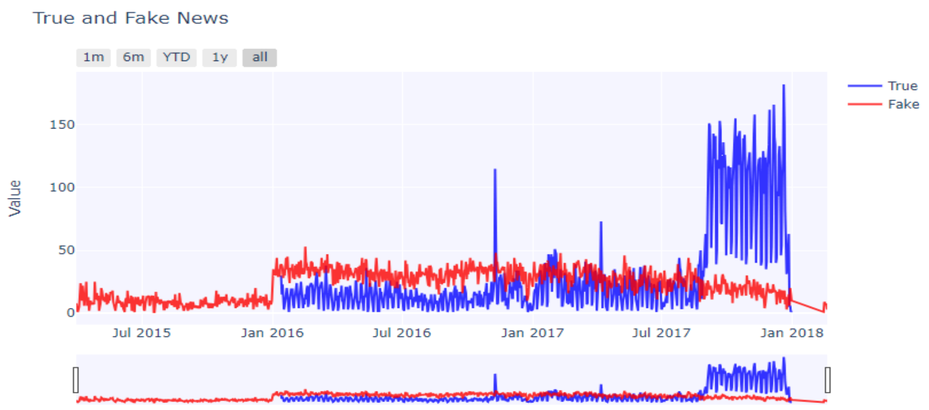


Insights:

- True news doesn't involve much trump instead on Republican Party and Russia
- There are news about Budget,military which comes under government news

Time series analysis- Fake/True news

```
#Creating the count of output based on date
fake=fake_news.groupby(['date'])['output'].count()
fake=pd.DataFrame(fake)

true=true_news.groupby(['date'])['output'].count()
true=pd.DataFrame(true)

#Plotting the time series graph
fig = go.Figure()
fig.add_trace(go.Scatter(
         x=true.index,
         y=true['output'],
         name='True',
    line=dict(color='blue'),
    opacity=0.8))

fig.add_trace(go.Scatter(
         x=fake.index,
         y=fake['output'],
         name='Fake',
    line=dict(color='red'),
    opacity=0.8))

fig.update_xaxes(
    rangeslider_visible=True,
    rangeselector=dict(
        buttons=list([
            dict(count=1, label="1m", step="month", stepmode="backward"),
            dict(count=6, label="6m", step="month", stepmode="backward"),
            dict(count=1, label="YTD", step="year", stepmode="todate"),
            dict(count=1, label="1y", step="year", stepmode="backward"),
            dict(step="all")
        ])
    )
)
fig.update_layout(title_text='True and Fake News',plot_bgcolor='rgb(248, 248, 255)',yaxis_title='Value')
fig.show()
```



Insights:

- True news got their dominance since Aug 2017. As they are seen at a very higher rates.That is a good sign
- There are few outliers in true news where it was higher than the fake news(Nov 9, 2016 and Apr 7, 2017)
- Our dataset has more fake news than the true one as we can see that we don't have true news data for whole 2015, So the fake news classification will be pretty accurate than the true news getting classified

Stemming & Vectorizing

Stemming the reviews:

Stemming is a method of deriving root word from the inflected word. Here we extract the reviews and convert the words in reviews to its root word. for example,

- Going->go
- Finally->fina

```
#Extracting 'reviews' for processing
news_features=clean_news.copy()
news_features=news_features[['news']].reset_index(drop=True)
news_features.head()
```

	news
0	donald trump sends embarrassing new year's eve...
1	drunk bragging trump staffer started russian c...
2	sheriff david clarke becomes internet joke thr...
3	trump obsessed even obama's name coded website...
4	pope francis called donald trump christmas spe...

```
stop_words = set(stopwords.words("english"))
#Performing stemming on the review dataframe
ps = PorterStemmer()

#splitting and adding the stemmed words except stopwords
corpus = []
for i in range(0, len(news_features)):
    news = re.sub('[^a-zA-Z]', ' ', news_features['news'][i])
    news= news.lower()
    news = news.split()
    news = [ps.stem(word) for word in news if not word in stop_words]
    news = ' '.join(news)
    corpus.append(news)

corpus[1]
```

```
'drunk brag trump staffer start russian collus investigationhous intellig committe chairman devin nune go bad day assumpt like mani us christoph steeledossi prompt russia investig lash depart justic fbi or
der protect trump happen dossier start investig accord document obtain new york timesform trump campaign advis georg papadopoulo drunk wine bar reveal knowledg russian opposit research hillari clintonon tc
p papadopoulo covfef boy trump administr alleg much larger role none damn drunken fool wine bar coffe boy help arrang new york meet trump presid abdel fattah elsisi egypt two month elect known former aid s
et meet world leader trump team trump ran mere coffe boyin may papadopoulo reveal australian diplomat alexand downer russian offici shop around possibl dirt thendemocrat presidenti nomine hillari clinton e
xactli much mr papadopoulo said night kensington wine room australian alexand downer unclear report state two month later leak democrat email began appear onlin australian offici pa...'
```

This is how a line looks like now, as computer cannot understand words and their sentiment we need to convert these words into 1's and 0's. To encode it we use TFIDF.

TFIDF(Term Frequency — Inverse Document Frequency)

TF-IDF stands for "Term Frequency — Inverse Document Frequency". This is a technique to quantify a word in documents, we generally compute a weight to each word which signifies the importance of the word in the document and corpus. This method is a widely used technique in Information Retrieval and Text Mining.

Here we are splitting as bigram (two words) and consider their combined weight.Also we are taking only the top 5000 words from the news.

```
tfidf_vectorizer = TfidfVectorizer(max_features=5000,ngram_range=(2,2))
# TF-IDF feature matrix
X= tfidf_vectorizer.fit_transform(news_features['news'])
X.shape
```

```
(44888, 5000)
```

```
#Getting the target variable
y=clean_news['output']
```

Checking for balance of data:

We should be careful about when handling imbalance data. If it is imbalanced, the model will be biased towards the higher frequency class and returns max output.

```
print(f'Original dataset shape : {Counter(y)}')
```

```
Original dataset shape : Counter({0: 23471, 1: 21417})
```

Our dataset is nearly a balanced one. So let's leave balancing it.

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Confusion Matrix")
    else:
        print('Confusion Matrix')

    thresh = cm.max() / 2.
    for i in range (cm.shape[0]):
        for j in range (cm.shape[1]):
            plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

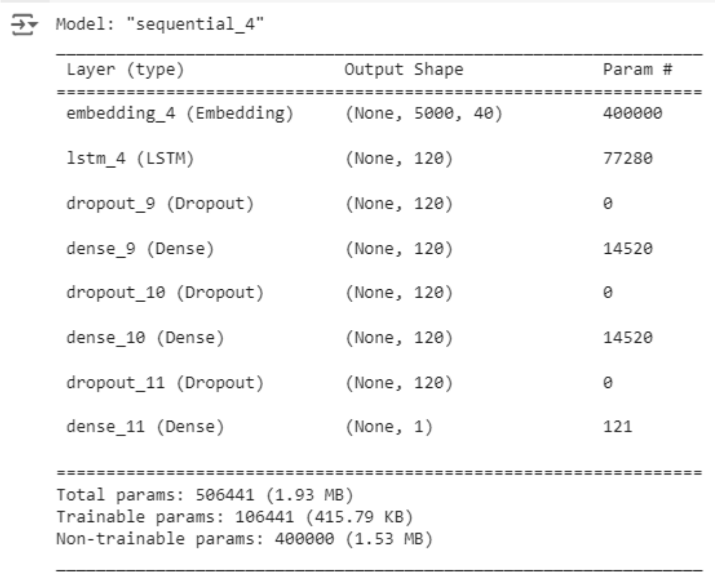
LSTM Architecture

We used a Long Short-Term Memory (LSTM) network, which is particularly adept at handling sequential data and capturing long-term dependencies. Our model architecture leverages pre-trained word embeddings, ensuring efficient and meaningful representation of words. The architecture includes multiple dense layers and dropout layers, strategically placed to enhance the model's learning capabilities and to mitigate overfitting. By incorporating callbacks such as Early Stopping and ReduceLROnPlateau, we ensure that the model training is both efficient and robust, dynamically adjusting learning parameters to achieve optimal performance. Below is the detailed code implementation of the LSTM model, followed by its compilation and training strategy:

```
lstm_model = Sequential([
    Embedding(voc_size, embedding_vector_features, input_length=sent_length, trainable=False),
    LSTM(120),
    Dropout(0.4),
    Dense(120, activation='relu'),  # Added Dense layer with 50 units
    Dropout(0.4),
    Dense(120, activation='relu'),  # Added Dense layer with 50 units
    Dropout(0.4),# Dropout layer with rate 0.4
    Dense(1, activation='sigmoid')  # Output layer
])

early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=5, min_lr=1e-5,
verbose=1)
```

```
lstm_model.compile(optimizer=tf.keras.optimizers.AdamW(learning_rate=0.001, weight_decay=0.0001),
                  loss=tf.keras.losses.BinaryCrossentropy(), metrics='accuracy')
```

```
lstm_model.summary()
```



```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 5000, 40)          400000

 lstm_4 (LSTM)               (None, 120)               77280

 dropout_9 (Dropout)         (None, 120)               0

 dense_9 (Dense)             (None, 120)               14520

 dropout_10 (Dropout)        (None, 120)               0

 dense_10 (Dense)            (None, 120)               14520

 dropout_11 (Dropout)        (None, 120)               0

 dense_11 (Dense)            (None, 1)                 121

=================================================================
Total params: 506441 (1.93 MB)
Trainable params: 106441 (415.79 KB)
Non-trainable params: 400000 (1.53 MB)
_________________________________________________________________
```

```
len(embedded_docs),y.shape
```

(44888, (44888,))

Fitting the LSTM Model

Before fitting to the model, let's consider the padded embedded object as X and y as y itself and convert them into an array.

```
# Converting the X and y as array
X_final=np.array(embedded_docs)
```

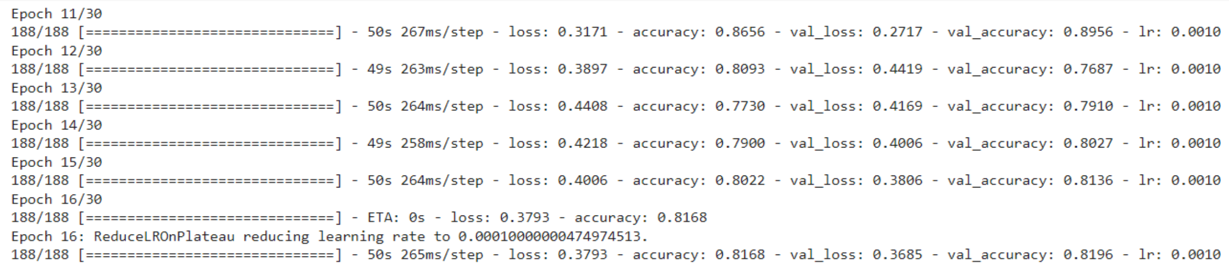
```
y_final=np.array(y)

#Check shape of X and y final
X_final.shape,y_final.shape
```
((44888, 5000), (44888,))

Let's split our new X and y variable into train and test and proceed with fitting the model to the data.

```
X_train, X_test, y_train, y_test = train_test_split(X_final, y_final, test_size=0.33, random_state=42)
```

We have considered 30 epochs and 128 as batch size.

```
history = lstm_model.fit(X_train, y_train, epochs=30, batch_size=80, validation_split=0.2,
callbacks=[early_stopping, reduce_lr])
```

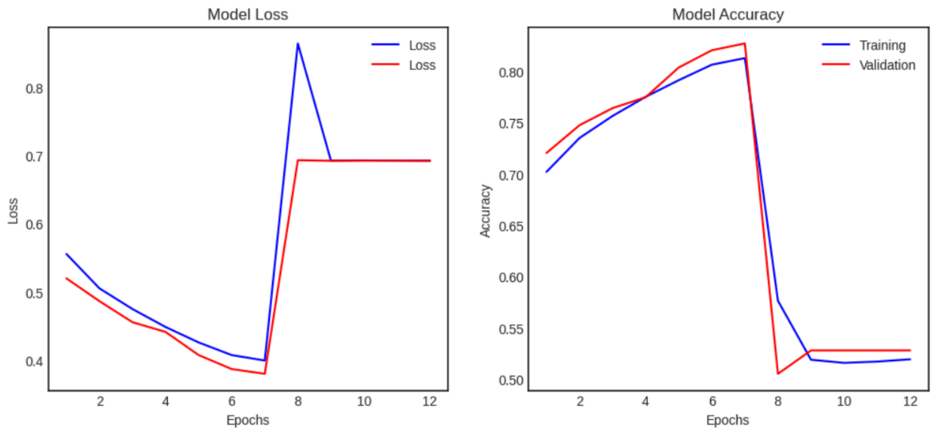


```
Epoch 11/30
188/188 [==============================] - 50s 267ms/step - loss: 0.3171 - accuracy: 0.8656 - val_loss: 0.2717 - val_accuracy: 0.8956 - lr: 0.0010
Epoch 12/30
188/188 [==============================] - 49s 263ms/step - loss: 0.3897 - accuracy: 0.8093 - val_loss: 0.4419 - val_accuracy: 0.7687 - lr: 0.0010
Epoch 13/30
188/188 [==============================] - 50s 264ms/step - loss: 0.4408 - accuracy: 0.7730 - val_loss: 0.4169 - val_accuracy: 0.7910 - lr: 0.0010
Epoch 14/30
188/188 [==============================] - 49s 258ms/step - loss: 0.4218 - accuracy: 0.7900 - val_loss: 0.4006 - val_accuracy: 0.8027 - lr: 0.0010
Epoch 15/30
188/188 [==============================] - 50s 264ms/step - loss: 0.4006 - accuracy: 0.8022 - val_loss: 0.3806 - val_accuracy: 0.8136 - lr: 0.0010
Epoch 16/30
188/188 [==============================] - ETA: 0s - loss: 0.3793 - accuracy: 0.8168
Epoch 16: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.
188/188 [==============================] - 50s 265ms/step - loss: 0.3793 - accuracy: 0.8168 - val_loss: 0.3685 - val_accuracy: 0.8196 - lr: 0.0010
```

Evaluation

Let's plot the accuracies and loss per epochs:

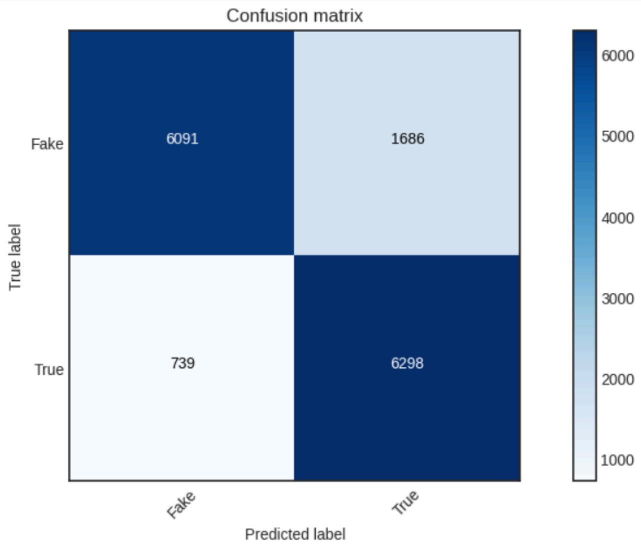
```
# Extract data
train_loss = history.history['loss']
val_loss = history.history['val_loss']
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
epochs = range(1, len(train_loss) + 1)

# Plotting loss
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(epochs, train_loss, 'b-', label='Loss')
plt.plot(epochs, val_loss, 'r-', label='Loss')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Plotting accuracy
plt.subplot(1, 2, 2)
plt.plot(epochs, train_acc, 'b-', label='Training')
plt.plot(epochs, val_acc, 'r-', label='Validation')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



Now, let's predict the output for our test data and evaluate the predicted values with y_test

```
y_pred = lstm_model.predict(X_test)
463/463 [==============================] - 32s 68ms/step
```

```
#Creating confusion matrix
cm = metrics.confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cm, classes=['Fake', 'True'])
```



```
#Checking for accuracy
accuracy_score(y_test,y_pred)
0.8363034966923181
# Creating classification report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.78      0.83      7777
           1       0.79      0.89      0.84      7037

    accuracy                           0.84     14814
   macro avg       0.84      0.84      0.84     14814
weighted avg       0.84      0.84      0.84     14814
```

BiDirectional LSTM

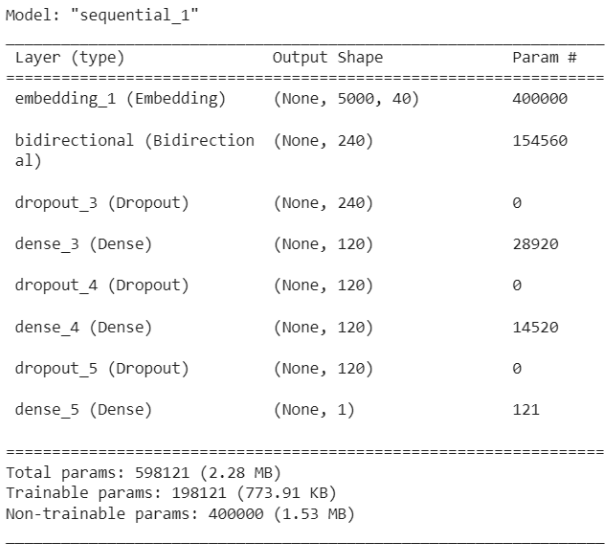
Bi-LSTM is an extension of normal LSTM with two independent RNN's together. The normal LSTM is uni directional where it cannot know the future words whereas in Bi-LSTM we can predict the future use of words as there is a backward information passed on from the other RNN layer in reverse.

There is only one change made in the code compared to the LSTM, here we use Bidirectional() function and call LSTM inside.

```
bilstm_model = Sequential([
    Embedding(voc_size, embedding_vector_features, input_length=sent_length, trainable=False),
    Bidirectional(LSTM(120)),
    Dropout(0.4),
    Dense(120, activation='relu'),  # Added Dense layer with 50 units
    Dropout(0.4),
    Dense(120, activation='relu'),  # Added Dense layer with 50 units
    Dropout(0.4),# Dropout layer with rate 0.4
    Dense(1, activation='sigmoid')  # Output layer
])

early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
reduce_lr  =  tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss',  factor=0.1,  patience=5,  min_lr=1e-5,
verbose=1)
```

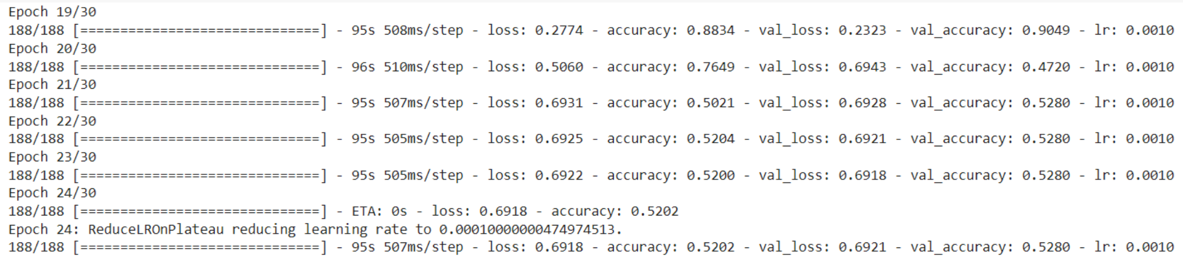
```
bilstm_model.compile(optimizer=tf.keras.optimizers.AdamW(learning_rate=0.001, weight_decay=0.0001),
                  loss=tf.keras.losses.BinaryCrossentropy(), metrics='accuracy')
```

```
bilstm_model.summary()
```



```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 5000, 40)          400000

 bidirectional (Bidirection  (None, 240)               154560
 al)

 dropout_3 (Dropout)         (None, 240)               0

 dense_3 (Dense)             (None, 120)               28920

 dropout_4 (Dropout)         (None, 120)               0

 dense_4 (Dense)             (None, 120)               14520

 dropout_5 (Dropout)         (None, 120)               0

 dense_5 (Dense)             (None, 1)                 121

=================================================================
Total params: 598121 (2.28 MB)
Trainable params: 198121 (773.91 KB)
Non-trainable params: 400000 (1.53 MB)
_________________________________________________________________
```

```
history1 = bilstm_model.fit(X_train, y_train, epochs=30, batch_size=80, validation_split=0.2,
callbacks=[early_stopping, reduce_lr])
```

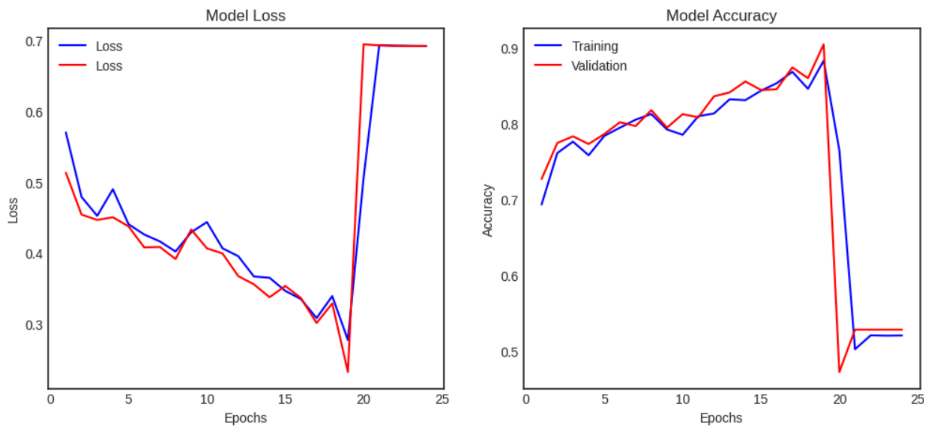


```
Epoch 19/30
188/188 [==============================] - 95s 508ms/step - loss: 0.2774 - accuracy: 0.8834 - val_loss: 0.2323 - val_accuracy: 0.9049 - lr: 0.0010
Epoch 20/30
188/188 [==============================] - 96s 510ms/step - loss: 0.5060 - accuracy: 0.7649 - val_loss: 0.6943 - val_accuracy: 0.4720 - lr: 0.0010
Epoch 21/30
188/188 [==============================] - 95s 507ms/step - loss: 0.6931 - accuracy: 0.5021 - val_loss: 0.6928 - val_accuracy: 0.5280 - lr: 0.0010
Epoch 22/30
188/188 [==============================] - 95s 505ms/step - loss: 0.6925 - accuracy: 0.5204 - val_loss: 0.6921 - val_accuracy: 0.5280 - lr: 0.0010
Epoch 23/30
188/188 [==============================] - 95s 505ms/step - loss: 0.6922 - accuracy: 0.5200 - val_loss: 0.6918 - val_accuracy: 0.5280 - lr: 0.0010
Epoch 24/30
188/188 [==============================] - ETA: 0s - loss: 0.6918 - accuracy: 0.5202
Epoch 24: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.
188/188 [==============================] - 95s 507ms/step - loss: 0.6918 - accuracy: 0.5202 - val_loss: 0.6921 - val_accuracy: 0.5280 - lr: 0.0010
```

Evaluation of Model

```
# Extract data
train_loss = history1.history['loss']
val_loss = history1.history['val_loss']
train_acc = history1.history['accuracy']
val_acc = history1.history['val_accuracy']
epochs = range(1, len(train_loss) + 1)
```

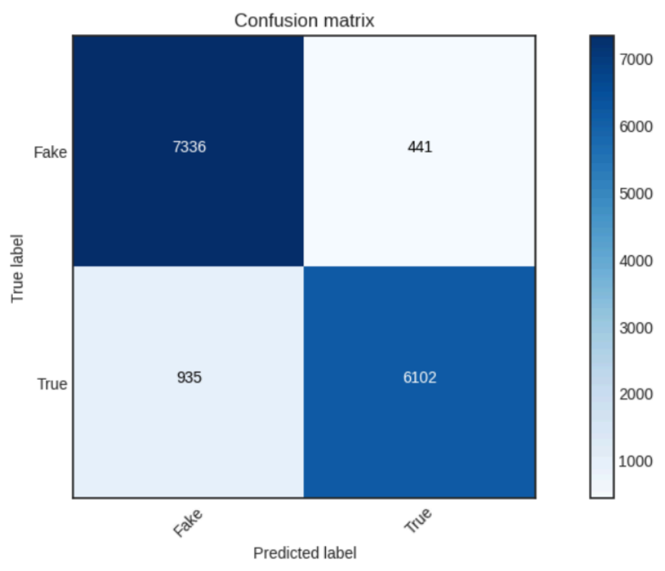
```python
# Plotting loss
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(epochs, train_loss, 'b-', label='Loss')
plt.plot(epochs, val_loss, 'r-', label='Loss')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Plotting accuracy
plt.subplot(1, 2, 2)
plt.plot(epochs, train_acc, 'b-', label='Training')
plt.plot(epochs, val_acc, 'r-', label='Validation')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



```python
y_pred1=bilstm_model.predict(X_test)
```

```
463/463 [==============================] - 54s 114ms/step
```

```python
#Confusion matrix
cm = metrics.confusion_matrix(y_test, y_pred1)
plot_confusion_matrix(cm,classes=['Fake','True'])
```



```
#Calculating Accuracy score
accuracy_score(y_test,y_pred1)
0.9071148913190226
# Creating classification report
print(classification_report(y_test,y_pred1))
```

```
              precision    recall  f1-score   support

           0       0.89      0.94      0.91      7777
           1       0.93      0.87      0.90      7037

    accuracy                           0.91     14814
   macro avg       0.91      0.91      0.91     14814
weighted avg       0.91      0.91      0.91     14814
```