FAKE NEWS DETECTION USING NLP

NAME : KAMALESHWARAN R

NM ID :aut61772221T304

REG NO:61772221T304

PROBLEMDEFINITION

The problem is to develop a Fake News Detection Model, which is able to distinguish between a Genuine news and Fake news and To develop the model we use the Kaggle dataset. Natural Language processing (NLP) is basically How you can Teach machine to understand Human languages and extract Meaning from text.

1. **Data Collection and Preprocessing:**

The Kaggle dataset :

Trump Is So Obsessed He Even Has Obama's Name Coded Into His Website (IMAGES)	On Christmas day, Donald Trump announced that he would be back to work the following day, but he i...	News	December 29, 2017
Pope Francis Just Called Out Donald Trump During His Christmas Speech	Pope Francis used his annual Christmas Day message to rebuke Donald Trump without	News	December 25, 2017

- Acquire a diverse dataset of news articles, each labeled as either authentic or fake.
- Preprocess the text data by cleaning, tokenizing, and converting it into a structured format suitable for NLP analysis.

2. **Feature Extraction:**

- Extract relevant features from the preprocessed text, such as word embeddings, TF-IDF scores, or other linguistic and semantic features, to represent the content in a numerical format.

3. **Model Training and Evaluation:**

1. Train a machine learning or deep learning model using the preprocessed and feature-extracted data, employing an appropriate algorithm such as Support Vector Machines, Random Forest, recurrent neural networks (RNNs), or transformers (e.g., BERT, GPT).
2. Evaluate the model's performance using appropriate metrics (e.g., accuracy, precision, recall, F1-score) on a separate validation or test dataset to assess its ability to discriminate between genuine and fake news.

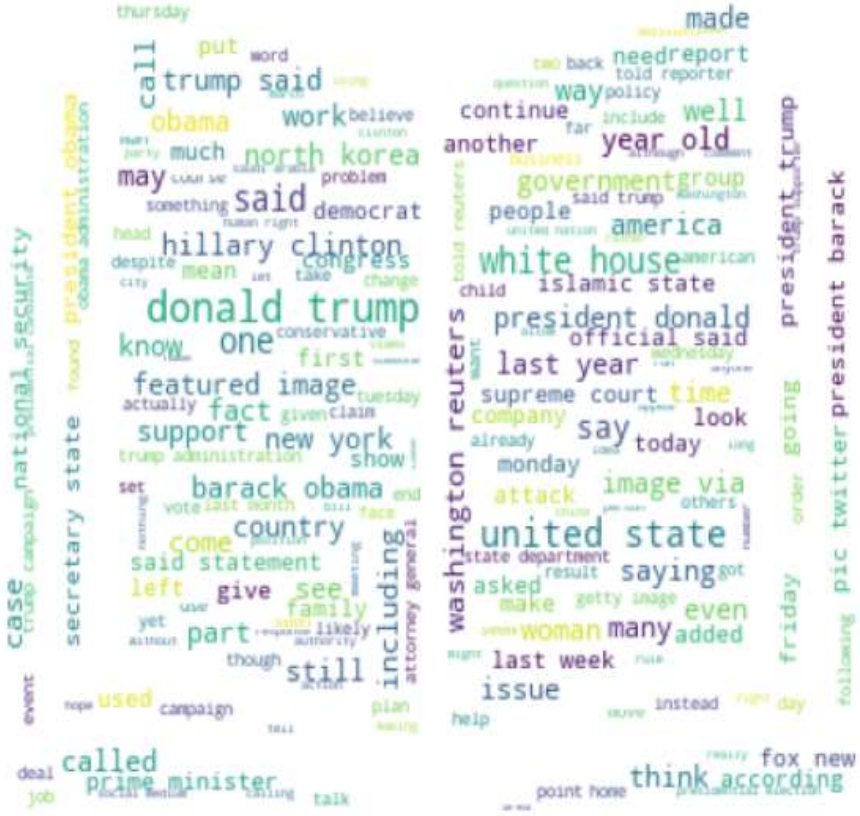
4. **Fine-Tuning and Optimization:**

- Fine-tune the model by adjusting hyperparameters, exploring different architectures, or optimizing the training process to achieve better performance.

5. **Real-Time Prediction:**

- Implement the trained model for real-time prediction, allowing users to input new text or articles and receive predictions regarding their authenticity.

The ultimate aim is to create a reliable and efficient f



INTRODUCTION :

To identify news, sentiment analysis using NLP can be an effective strategy. NLP algorithms can ascertain the intention and any biases of an author by analyzing the emotions displayed in a news story or social media post.

Techniques of NLP and Machine learning can be used to create models which can help to detect fake news. In this paper we have presented six LSTM models using the techniques of NLP and ML. The datasets in comma-separated values format, pertaining to political domain were used in the project.

NLP USING TECHNIQUES

- Keyword Analysis
- Sentiment Analysis
- Language Models
- Named Entity Recognition
- Clickbait Analysis
- Stance Detection
- Duplication Detection
- Source Reputation

ADVANCED MACHINE LEARING TECHNIQUES

- Natural Language Processing (NLP)
- Sentiment Analysis
- Fact-Checking Bots
- Source Reputation Analysis
- Cross-Referencing and Verification
- User Behavior Analysis
- Metadata Analysis
- Crowdsourced Fact-Checking

DATASET :

Donald Trump Sends Out Embarrassing New Year's Eve Message; This is Disturbing	Donald Trump just couldn t wish all Americans a Happy New Year and leave it at that. Instead, he had...	News	December 31, 2017
Drunk Bragging Trump Staffer Started Russian Collusion Investigation	House Intelligence Committee Chairman Devin Nunes is going to have a bad day. He s been under the as...	News	December 31, 2017

The above dataset is the "Fake News dataset" which has data from articles . We are going to preprocess the data and show them.

DEEP LEARNING : (CNNs & RNNs)

Deep Learning is a subset of machine learning that involves the use of artificial neural networks with multiple layers, allowing it to analyze and recognize complex patterns in data. In the context of fake news detection

Deep Neural Networks: These are algorithms that consist of multiple layers of interconnected nodes (neurons), enabling them to learn hierarchical representations of data. Each layer processes the input data and passes it to the next layer for further abstraction and analysis.

Detection of Manipulation or Fabrication: Deep learning models can be trained on large datasets of genuine and manipulated media content. By learning from these datasets, they can identify subtle visual or temporal cues that indicate alterations, enhancements, or fabrications in images and videos

CNNs

Convolutional Neural Networks (CNNs), also known as ConvNets, are a class of deep neural networks primarily designed for processing and analyzing visual data, such as images and videos. They are a specialized type of artificial neural network architecture that excels at tasks involving pattern recognition within grids of data, like the pixel values in an image.

CNNs are widely used in computer vision tasks, including image classification, object detection, image segmentation, and facial recognition. They have also been applied in various fields beyond vision, such as natural language processing and audio analysis, when dealing with grid-like data structures.

CNNs have played a pivotal role in advancing the accuracy and performance of machine learning models for visual data analysis, making them a foundational technology in many applications involving images and videos.

- Pooling Layers
- Fully Connected Layers
- Local Connectivity

RNNs

Recurrent Neural Networks, are a type of artificial neural network designed for sequential data processing. Unlike traditional feedforward

neural networks, RNNs have connections that loop back on themselves, allowing them to maintain a hidden state or memory of previous inputs. This makes RNNs well-suited for tasks involving sequences, such as natural language processing, speech recognition, and time series prediction. However, they suffer from vanishing gradient problems, which can limit their ability to capture long-range dependencies in data. More advanced variations of RNNs, like LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit), have been developed to address these issues.

PROGRAM:

```
import pandas as pd
not_fake = pd.read_csv("C:\\Users\\ADMIN\\Documents\\True.csv")
fake =
pd.read_csv("C:\\Users\\ADMIN\\Documents\\Fake.csv",low_memory=
False)
X = not_fake["title"].tolist() + fake["title"].tolist()
y = [0] * len(not_fake) + [1] * len(fake)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
!pip install simple_nlp_library
from simple_nlp_library import preprocessing, embeddings
stop_words = preprocessing.stop_words()
vectors = embeddings.vectors()
```

```
X_train_vec = [embeddings.tokens_vector(vectors,
preprocessing.semantic_tokens(stop_words, x)) for x in X_train]

X_test_vec = [embeddings.tokens_vector(vectors,
preprocessing.semantic_tokens(stop_words, x)) for x in X_test]

from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(hidden_layer_sizes=(25), early_stopping=True)

clf.fit(X_train_vec, y_train)

import matplotlib.pyplot as plt

plt.plot(clf.validation_scores_)

from sklearn.metrics import accuracy_score

accuracy_score(y_train, clf.predict(X_train_vec))

accuracy_score(y_test, clf.predict(X_test_vec))

accuracy_score(y_test, clf.predict(X_test_vec))
```

OUTPUT :

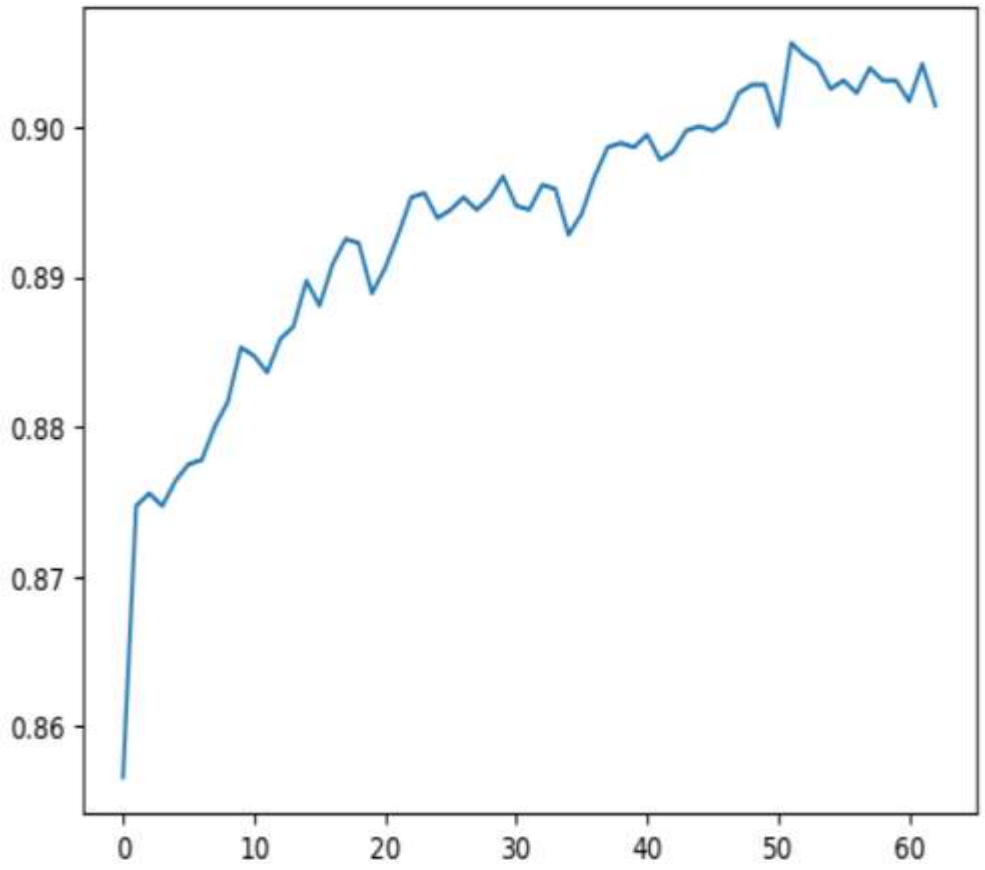
Out[11]:

MLPClassifier

MLPClassifier(early_stopping=True, hidden_layer_sizes=25)

0.8847951914514692

Sample Output:



In The above program We Import the Pandas Library file,We are having two sets of dataset i.e the fake news and true news data set ,we read those data using two different variables.we preprocess the data using the inbuilt functions.

NOTEBOOK COPY OF THE PROGRAM

[ ]:
```
import pandas as pd
not_fake = pd.read_csv("../input/fake-and-real-news-dataset/True.csv")
fake = pd.read_csv("../input/fake-and-real-news-dataset/Fake.csv")
X = not_fake["title"].tolist() + fake["title"].tolist()
y = [0] * len(not_fake) + [1] * len(fake)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
!pip install simple_nlp_library
from simple_nlp_library import preprocessing, embeddings
stop_words = preprocessing.stop_words()
vectors = embeddings.vectors()
X_train_vec = [embeddings.tokens_vector(vectors, preprocessing.semantic_tokens(stop_words, x)) for x in X_train]
X_test_vec = [embeddings.tokens_vector(vectors, preprocessing.semantic_tokens(stop_words, x)) for x in X_test]
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(hidden_layer_sizes=(25), early_stopping=True)
clf.fit(X_train_vec, y_train)
import matplotlib.pyplot as plt
plt.plot(clf.validation_scores_)
from sklearn.metrics import accuracy_score
accuracy_score(y_train, clf.predict(X_train_vec))
0.9032796926332202
accuracy_score(y_test, clf.predict(X_test_vec)
```

DATA IMPORTING:

```
# Fake news detection by NLP
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import nltk
import re
import string

from bs4 import BeautifulSoup
from nltk.corpus import stopwords

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

import keras
from keras.preprocessing import text,sequence
from keras.models import Sequential
from keras.layers import Dense,Embedding,LSTM,Dropout

import warnings
warnings.filterwarnings('ignore')
```

```
# DATA IMPORT
```

In [2]:
```
#data importing
real_news = pd.read_csv('True_News.csv')
fake_news = pd.read_csv('Fake_News.csv')
#here we import the data set of both the true and fake news
```

Reading CSV Files: Here the pd.read_csv() function from Pandas to read data from CSV files. CSV (Comma-Separated Values) files are a common data format for storing structured data.

- real_news = pd.read_csv('True_News.csv'): This line reads the data from a CSV file named 'True_News.csv' and stores it in a Pandas Data Frame ca led real_news. This file is expected to contain data related to "true" or real news articles.
- fake_news = pd.read_csv('Fake_News.csv'): Similarly, this line reads data from a CSV file named 'Fake_News.csv' and stores it in another Pandas Data Frame called fake news. This file is expected to contain data related to "fake" news articles.

DATA CLEANING:

del data['title']: This removes the 'title' column from the Data Frame. It's removed from the Data Frame's structure, and you will no longer be able to access this column's data.

Similarly, this line removes the 'subject' column from the Data Frame. This line removes the 'date' column from the Data Frame.

After running these lines, the Data Frame data wi l no longer contain the 'title', 'subject', and 'date' columns. This is often done when we want to focus on specific columns or when those columns are no longer needed for the plan to perform.

The code you provided contains a series of functions and a final function called cleaning that are used to preprocess text data in a Pandas Data Frame. The purpose of this preprocessing is to clean and prepare the text for various natural language processing (NLP) tasks, such as text classification or sentiment analysis.

```
# DATA CLEANING
```

```
In [10]: #the colums of title,subject,date are deleted as they are not used.
data['text']= data['subject'] + " " + data['title'] + " " + data['text']
del data['title']
del data['subject']
del data['date']
data.head()
```

Out[10]:

	text	target
0	politicsNews As U.S. budget fight looms, Repub...	0
1	politicsNews U.S. military to accept transgend...	0
2	politicsNews Senior U.S. Republican senator: '...	0
3	politicsNews FBI Russia probe helped by Austra...	0
4	politicsNews Trump wants Postal Service to cha...	0

This function combines the previous functions to create a comprehensive cleaning process. It takes a text as input and applies the following steps in order:

- Removes HTML content.
- Removes punctuation marks and special characters.
- Removes non-alphabet characters.

- Removes stop words and lemmatizes the remaining words.
- Returns the cleaned text.

In [22]:
```
#Removal of HTML Contents
def remove_html(text):
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

#Removal of Punctuation Marks
def remove_punctuations(text):
    return re.sub('\[[^]]*\]', '', text)

# Removal of Special Characters
def remove_characters(text):
    return re.sub("[^a-zA-Z]"," ",text)

#Removal of stopwords
def remove_stopwords_and_lemmatization(text):
    final_text = []
    text = text.lower()
    text = nltk.word_tokenize(text)

    for word in text:
        if word not in set(stopwords.words('english')):
            lemma = nltk.WordNetLemmatizer()
            word = lemma.lemmatize(word)
            final_text.append(word)
    return " ".join(final_text)

#Total function
def cleaning(text):
    text = remove_html(text)
    text = remove_punctuations(text)
    text = remove_characters(text)
    text = remove_stopwords_and_lemmatization(text)
    return text

#Apply function on text column
data['text']=data['text'].apply(cleaning)
```

DATA VISUALIZATION:

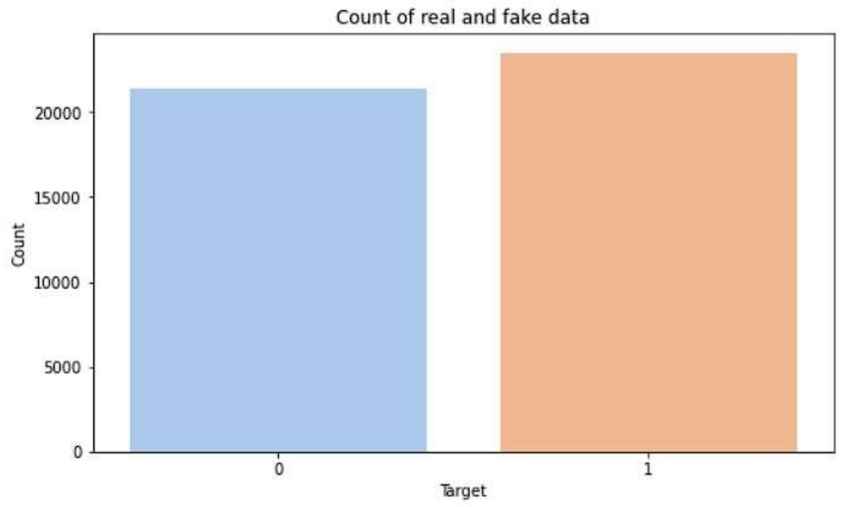
This prints the counts of each unique value in the "target" column. It will show you how many instances belong to each class, which is useful for understanding the class distribution.

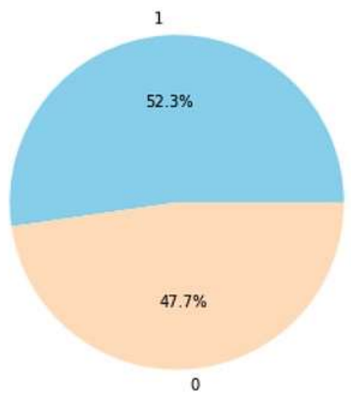
This code creates a pie chart in the second subplot. It uses the values and labels from the count of "target" classes to create the chart.

DATA VISUALIZATION

In [8]:
```
#we express the dataset of both true and fake in terms of a bar chart and a pie chart.
print(data["target"].value_counts())
fig, ax = plt.subplots(1,2, figsize=(19, 5))
g1 = sns.countplot(data.target,ax=ax[0],palette="pastel");
g1.set_title("Count of real and fake data")
g1.set_ylabel("Count")
g1.set_xlabel("Target")
g2 = plt.pie(data["target"].value_counts().values,explode=[0,0],labels=data.target.value_counts().index, autopct='%1.1f%%',colors
fig.show()
```

```
1    23481
0    21417
Name: target, dtype: int64
```





- X ="subject": This sets the values on the x-axis to come from the "subject" column, which represents the different subjects.
- Hue ='target': It adds color differentiation based on the "target" column. The plot wi l have two bars (real and fake) for each subject, with dif erent colors.

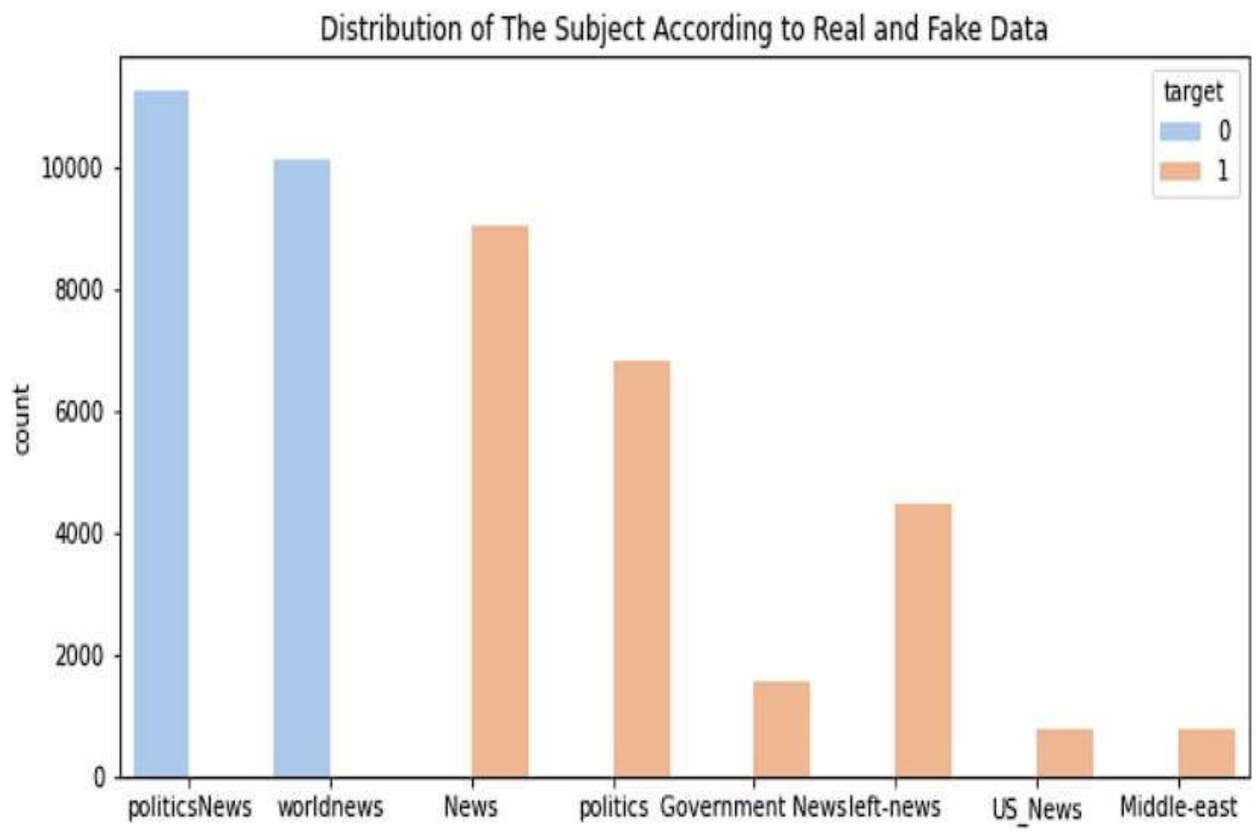
- Data =data: Specifies the DataFrame from which the data for the plot is retrieved.
- Palette ="pastel": Sets the color palette to "pastel," giving the plot a more colorful and visua ly appealing style.

```
In [9]: #here we get to see the contents of the subject and the count of news in each subject
        print(data.subject.value_counts())
        plt.figure(figsize=(10, 5))

        ax = sns.countplot(x="subject",  hue='target', data=data, palette="pastel")
        plt.title("Distribution of The Subject According to Real and Fake Data")
```

```
politicsNews       11272
worldnews          10145
News                9050
politics            6841
left-news           4459
Government News     1570
US_News              783
Middle-east          778
Name: subject, dtype: int64
```

```
Out[9]: Text(0.5, 1.0, 'Distribution of The Subject According to Real and Fake Data')
```



WORD CLOUD:

This code imports the Word Cloud class from the word cloud library and the STOPWORDS set, which contains common English stop words.

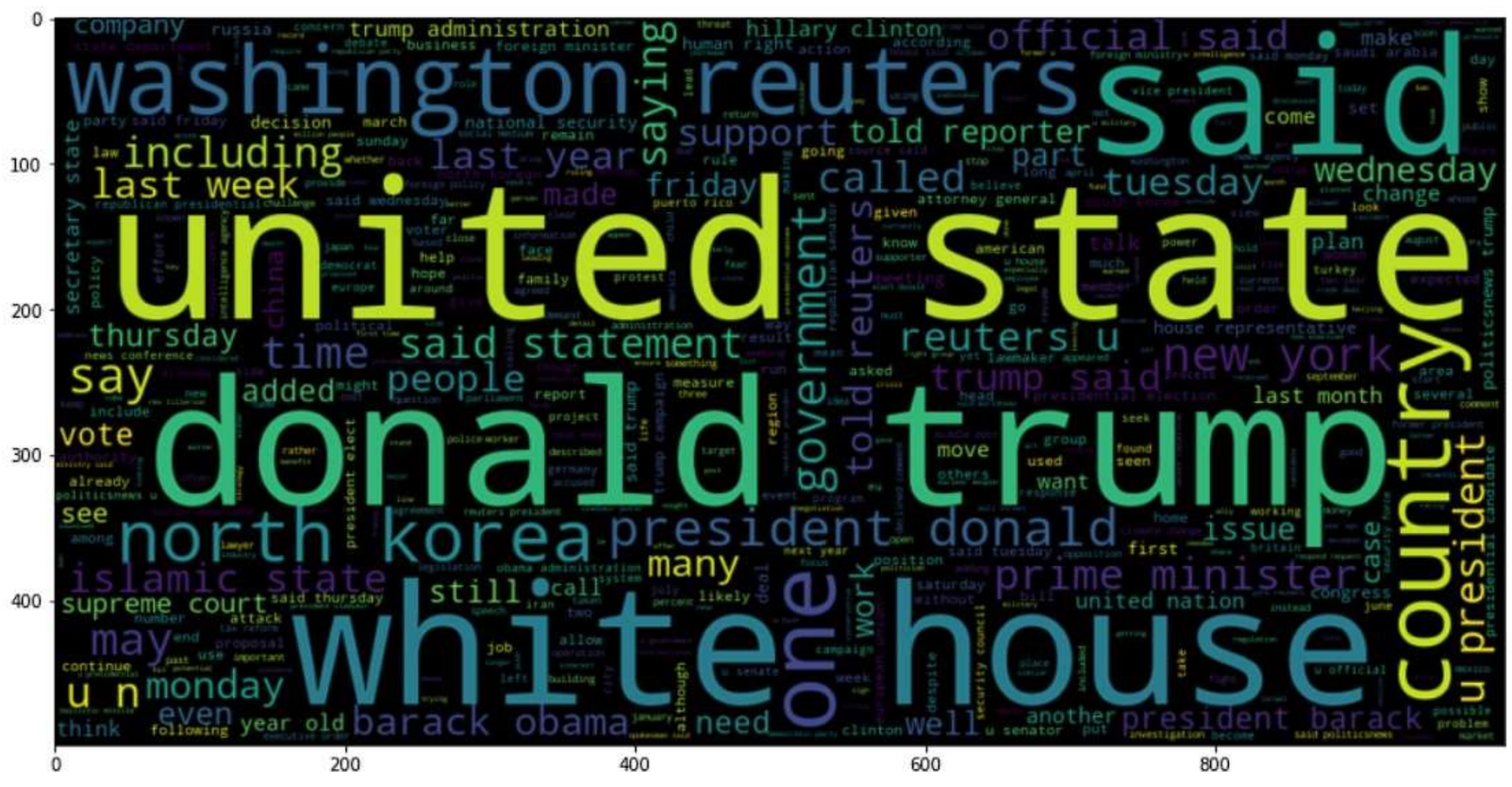
- Maximum number of words to include in the word cloud is set.
- Width and height specify the dimensions of the word cloud image.

REAL DATASET

 # 1. real news

In [26]:
```
from wordcloud import WordCloud,STOPWORDS
plt.figure(figsize = (15,15))
wc = WordCloud(max_words = 500 , width = 1000 , height = 500 , stopwords = STOPWORDS).generate(" ".join(data[data.target == 0]
plt.imshow(wc , interpolation = 'bilinear')
```

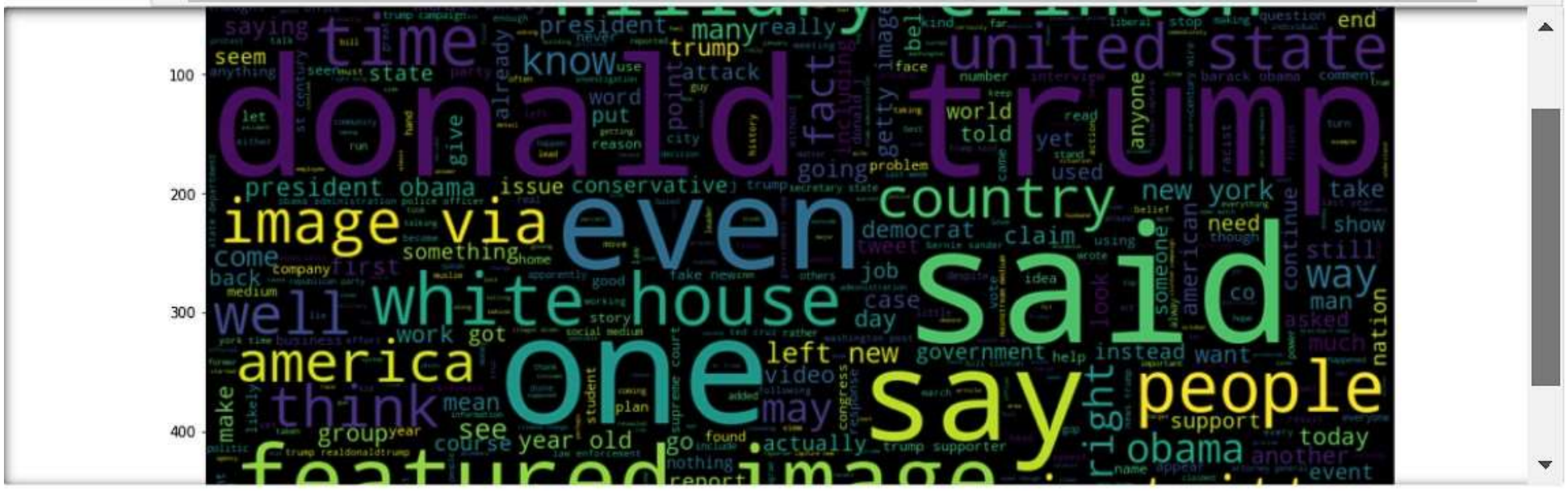
Out[26]: <matplotlib.figure.AxesImage at 0x1f000997310>



FAKE DATASET :

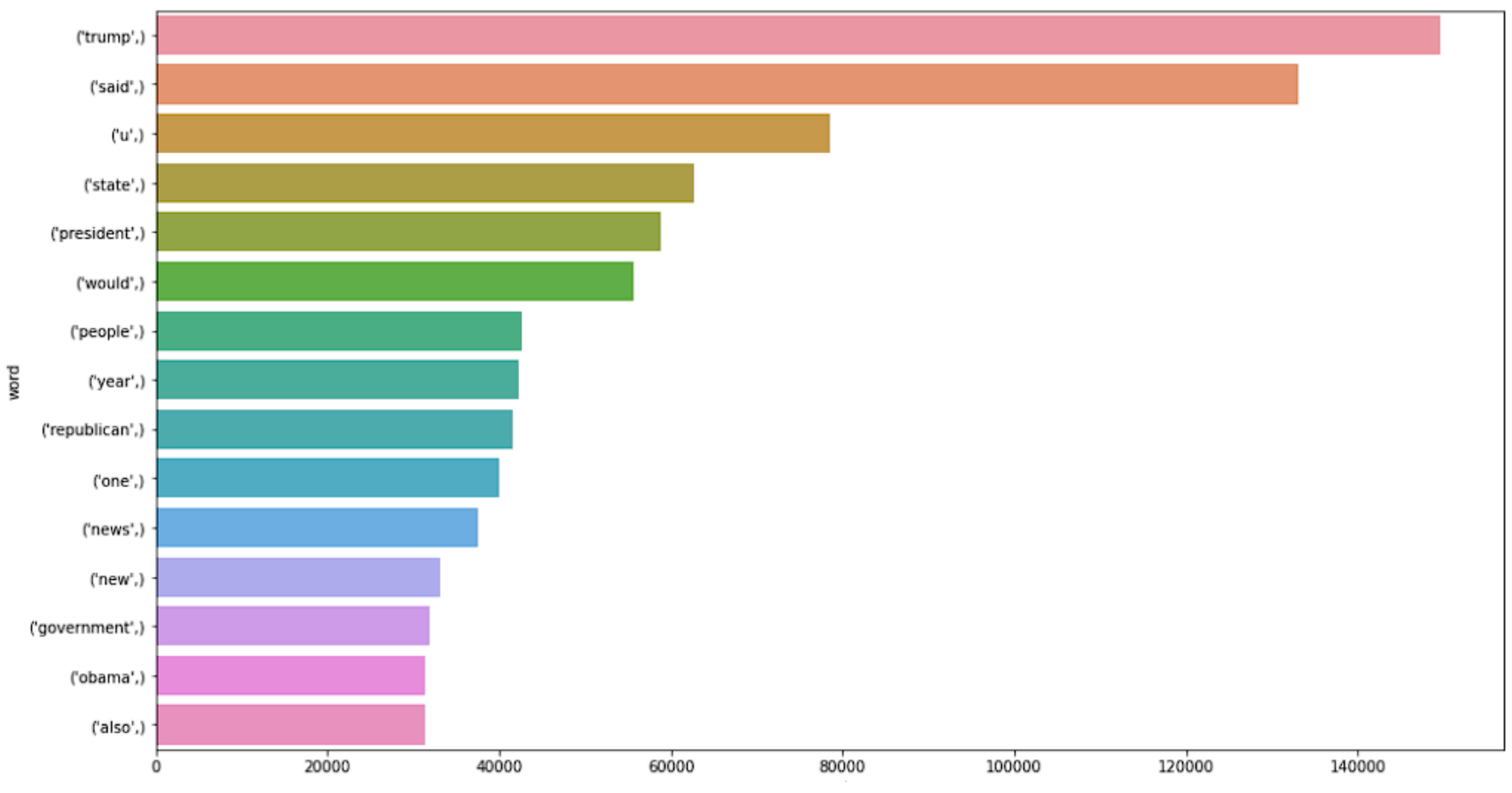
2. fake news

In [27]:
```
#the word cloud is used to visualize the most used words.
from wordcloud import WordCloud,STOPWORDS
plt.figure(figsize = (15,15))
wc = WordCloud(max_words = 500 , width = 1000 , height = 500 , stopwords = STOPWORDS).generate(" ".join(data[data.target == 1].te
plt.imshow(wc , interpolation = 'bilinear')
```



DATA ANA LYSIS:

Out[29]: <AxesSubplot:xlabel='count', ylabel='word'>



Finally, it creates a bar plot using Sea born to visualize the frequency of these n-grams, with the x-axis displaying the count and the y-axis showing the individual n-grams.

CONTENTS:

- ✓ **Model**
- ✓ **Splitting the data set**
- ✓ **Training**
- ✓ **Model Evaluation**

MODEL: LSTM

The model used here is LSTM. It is abbreviated as Long Short Term Memory. LSTM models, which are known as Long Short Term Memory models find use in detecting news due, to their strong capability to analyze and handle sequential data. This makes them highly suitable, for tasks involving the classification of text based content.

It's important to note that while LSTMs are powerful for modeling sequential data, the success of a fake news detection system often relies on the quality and diversity of the data used for training, as well as the design of the overall architecture, including the choice of features, data preprocessing, and post-processing steps. Moreover, fake news detection is a complex task, and no single model or

approach can guarantee perfect results, but LSTMs can be a valuable component of a larger solution.

SPLITTING OF DATASET:

The code you provided seems to be dividing a dataset into two parts; one, for training and the other for testing. Lets examine each component of this code and understand its purpose.

X_train and y_train; These variables are commonly used to represent the features (variables) and labels (target or dependent variable) of the training data respectively. In this case it appears that you're dealing with text data so X_train contains the text samples while y_train holds the target values or labels.

X_test and y_test; Similarly these variables represent the features and labels of the testing data. The testing data is utilized to assess how well your machine learning model performs after being trained on the training data.

This function randomly splits the input data into training and testing sets with a default ratio usually set at

75% for training and 25%, for testing.

```
# MODEL

In [33]: X_train, X_test, y_train, y_test = train_test_split(data['text'], data['target'], random_state=0)

In [34]: max_features = 10000
         maxlen = 300 #Length of news to 300

         tokenizer = text.Tokenizer(num_words=max_features)
         tokenizer.fit_on_texts(X_train)
         tokenized_train = tokenizer.texts_to_sequences(X_train)
         X_train = sequence.pad_sequences(tokenized_train, maxlen=maxlen)
```

This line defines a variable **max_features** and sets it to 10,000.

This variable **maxlen** sets the maximum length for each text sample.

A tokenizer is created using a text processing library, such, as Keras or Tensor Flow. The tokenizer object will only keep the 10,000 common words in its vocabulary during tokenization discarding less frequent words.

LSTM MODEL TRAINING:

The provided code defines a neural network model using the Keras library, which is commonly used for deep learning tasks.

model = Sequential():This initializes a sequential model. In a sequential model, layers are added one by one in a linear sequence, which is suitable for many standard deep learning architectures.

model.add(Embedding(max_features, output_dim=embed_size, input_length=maxlen, trainable=False): This adds an embedding layer to the model. The embedding layer is responsible for converting discrete text data into continuous vector representations. It uses pre-trained word embeddings with **output_dim** dimensions (set to 100), and the **max_features** specifies the vocabulary size. **trainable=False** indicates that these embeddings won't be fine-tuned during training, which is common when using pre-trained word embeddings .

model.add(LSTM(units=64, recurrent_dropout=0.1, dropout=0.1): The second LSTM layer has 64 units and doesn't return sequences, as it's the final recurrent layer in the network. Again, it includes regularization parameters for better generalization.

LSTM(long short term memory) MODEL

In [45]:
```
batch_size = 256
embed_size = 100
```

In [46]:
```
model = Sequential()
#embeddidng layer
model.add(Embedding(max_features, output_dim=embed_size, input_length=maxlen, trainable=False))
#LSTM
model.add(LSTM(units=128 , return_sequences = True , recurrent_dropout = 0.25 , dropout = 0.25))
model.add(LSTM(units=64 , recurrent_dropout = 0.1 , dropout = 0.1))
model.add(Dense(units = 32 , activation = 'relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer=keras.optimizers.Adam(lr = 0.01), loss='binary_crossentropy', metrics=['accuracy'])
```

model.compile(optimizer=keras.optimizers.Adam(lr =0.01), loss='binary_crossentropy', metrics=['accuracy']): This line compiles the model, specifying the optimizer , the loss function (binary cross-entropy, suitable for binary classification), and the evaluation metric (accuracy). This configuration prepares the model for training.

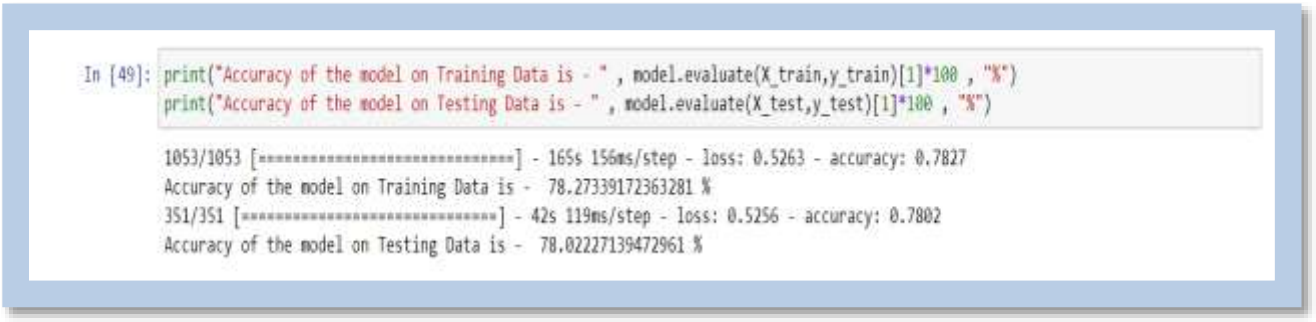
In [48]:
```
history = model.fit(X_train, y_train, validation_split=0.3, epochs=5, batch_size=batch_size, shuffle=True, verbose = 1)
```

```
Epoch 1/5
93/93 [==============================] - 729s 8s/step - loss: 0.5404 - accuracy: 0.7128 - val_loss: 0.3128 - val_accuracy: 0.86
82
Epoch 2/5
93/93 [==============================] - 726s 8s/step - loss: 0.3438 - accuracy: 0.8442 - val_loss: 0.3304 - val_accuracy: 0.84
56
Epoch 3/5
93/93 [==============================] - 721s 8s/step - loss: 0.2925 - accuracy: 0.8724 - val_loss: 0.2921 - val_accuracy: 0.88
22
Epoch 4/5
93/93 [==============================] - 744s 8s/step - loss: 0.2520 - accuracy: 0.8968 - val_loss: 0.2760 - val_accuracy: 0.90
81
Epoch 5/5
93/93 [==============================] - 784s 8s/step - loss: 0.2389 - accuracy: 0.9066 - val_loss: 0.5280 - val_accuracy: 0.77
55
```

The code trains the previously defined neural network model on the training data (**X_train** and **y_train**) for 5 epochs with a batch size of 256. It uses 30% of the training data for validation, shuffles the training data, and prints training progress information. The model's training history, including loss and accuracy, is stored in the **history** variable.

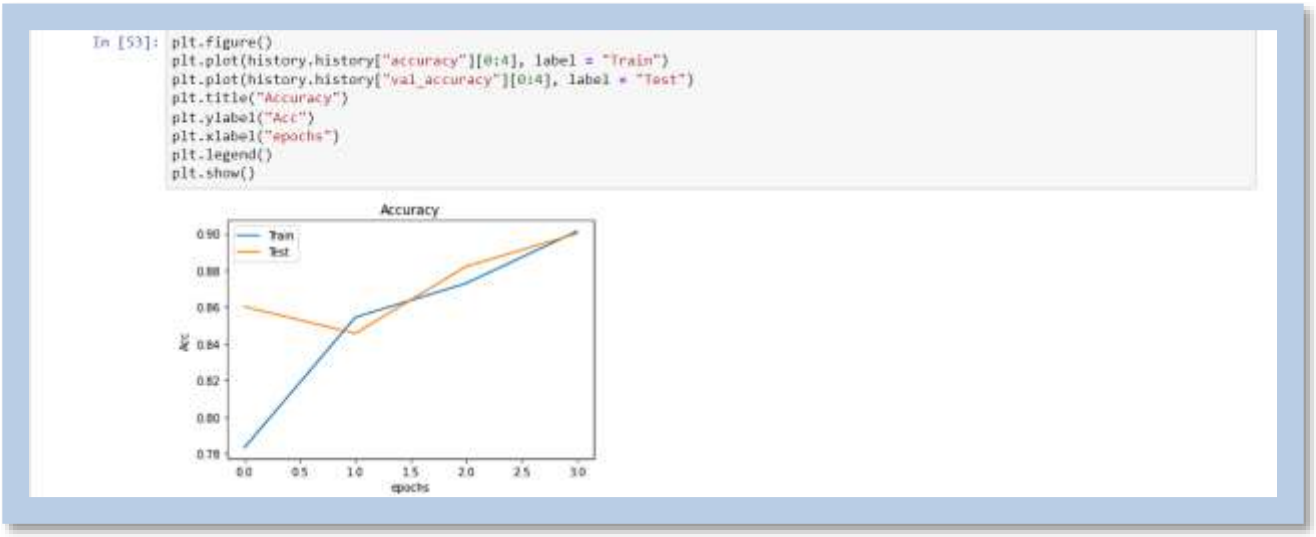
MODEL EVALUATION:

ACCURACY:



- In this code, a neural network model is trained using both the training and testing datasets, and the accuracy results are printed. The model demonstrates an accuracy of around 78.27% on the training data and 78.02% on the testing data.

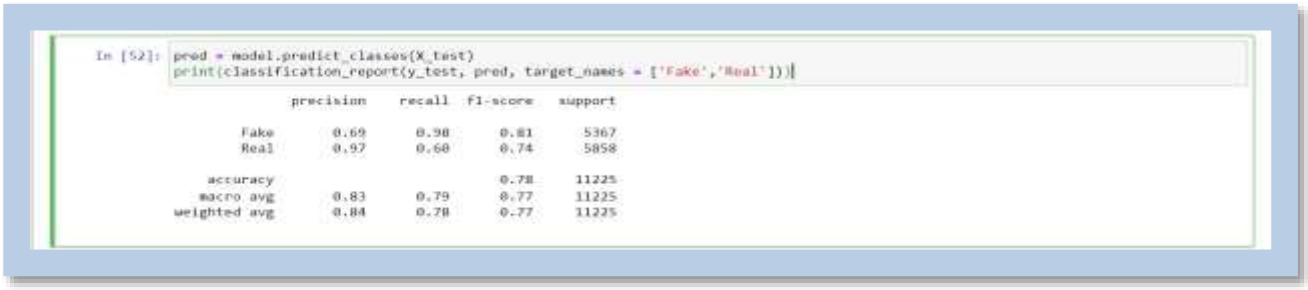
- This evaluation assesses the model's effectiveness in predicting the target variable, likely a binary classification task, based on the provided text data. The reported accuracy values serve as indicators of the model's ability to generalize its predictions to new and unseen data.



The legend distinguishes between the training and testing accuracy, and the x-axis represents the number of training epochs, while the y-axis represents accuracy.

The code snippet visualizes the training and testing accuracy trends, allowing us to observe how well the model performs as it trains.

CLASIFICATION REPORT:



```
In [52]: pred = model.predict_classes(X_test)
         print(classification_report(y_test, pred, target_names = ['Fake','Real']))

                      precision    recall  f1-score   support

                Fake       0.69      0.98      0.81      5367
                Real       0.97      0.60      0.74      5858

            accuracy                           0.78     11225
           macro avg       0.83      0.79      0.77     11225
        weighted avg       0.84      0.78      0.77     11225
```

With the help of the classification report, a comprehensive review of the model's effectiveness can be obtained. This includes a deep dive into metrics that are key to understanding its performance such as precision, recall, and the F1-score, and the scenario is applicable to each class, either 'Fake' or 'Real'. This undeniably forms an integral part of the overall assessment of the model's effectiveness especially, but certainly not limited to, a binary classification task.

CONCLUSION:

- Concluding this discussion, it becomes evident that the use of LSTM-based models in the quest to combat fake news holds great promise. LSTMs, with their innate ability to capture sequential patterns within textual data, are undeniably invaluable assets.

- In essence, while incorporating LSTM models can provide a potential roadmap for mitigating the issue of fake news, it's crucial to avoid viewing them as a sole solution for the time being. Only through the collective deployment of various systems and approaches can we be adequately prepared to wage a comprehensive and resilient battle against the spread of disinformation.