Fake News Analysis In Social Media Using IBM Watson

1. INTRODUCTION

1.1 Overview

Nowadays, fake news has become a common trend. Even trusted media houses are known to spread fake news and are losing their credibility. So, how can we trust any news to be real or fake? There should be a system that can analyze whether a given news post is fake or not. so the main of this project is to build an application that can analyze fake news

1.2 Purpose

In this project, we have built a classifier model that can identify news as real or fake. For this purpose, we have used data from Kaggle, but you can use any data to build this model following the same methods.

With the help of this project, you can create an NLP classifier to detect whether the news is real or fake.

2. LITERATURE SURVEY

2.1 Existing problem

In this day and age, it is extremely difficult to decide whether the news we come across is real or not. There are very few options to check the authenticity and all of them are sophisticated and not accessible to the average person. There is an acute need for a web-based fact-checking platform that harnesses the power of Machine Learning to provide us with that opportunity.

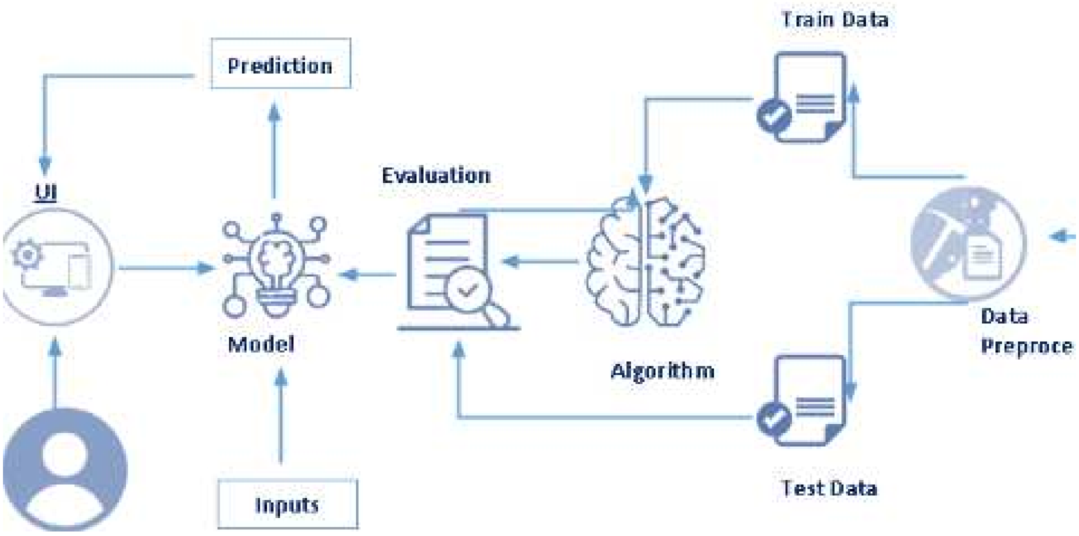
2.2 Proposed solution

In this project, we have built a classifier model that can identify news as real or fake. For this purpose, we have used data from Kaggle, but you can use any data to build this model following the same methods.

With the help of this project, you can create an NLP classifier to detect whether the news is real or fake.

3. THEORITICAL ANALYSIS

3.1 Block Diagram



3.2 Hardware / Software designing

Software Requirements:

- Anaconda Navigator

- Tensor flow
- Keras
- Flask

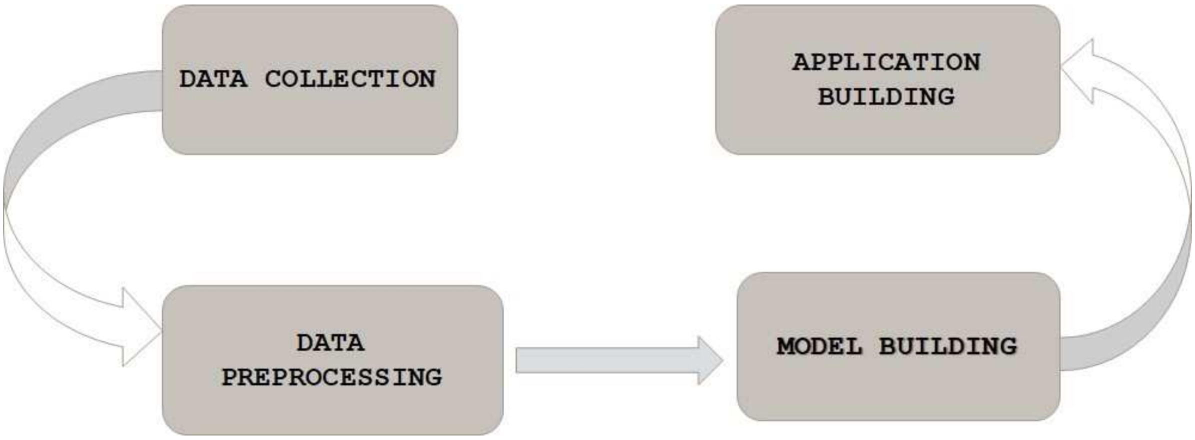
Hardware Requirements:

- Processor : Intel Core i3
- Hard Disk Space : Min 100 GB
- Ram : 4 GB
- Display : 14.1 “Color Monitor(LCD, CRT or LED)
- Clock Speed : 1.67 GHz

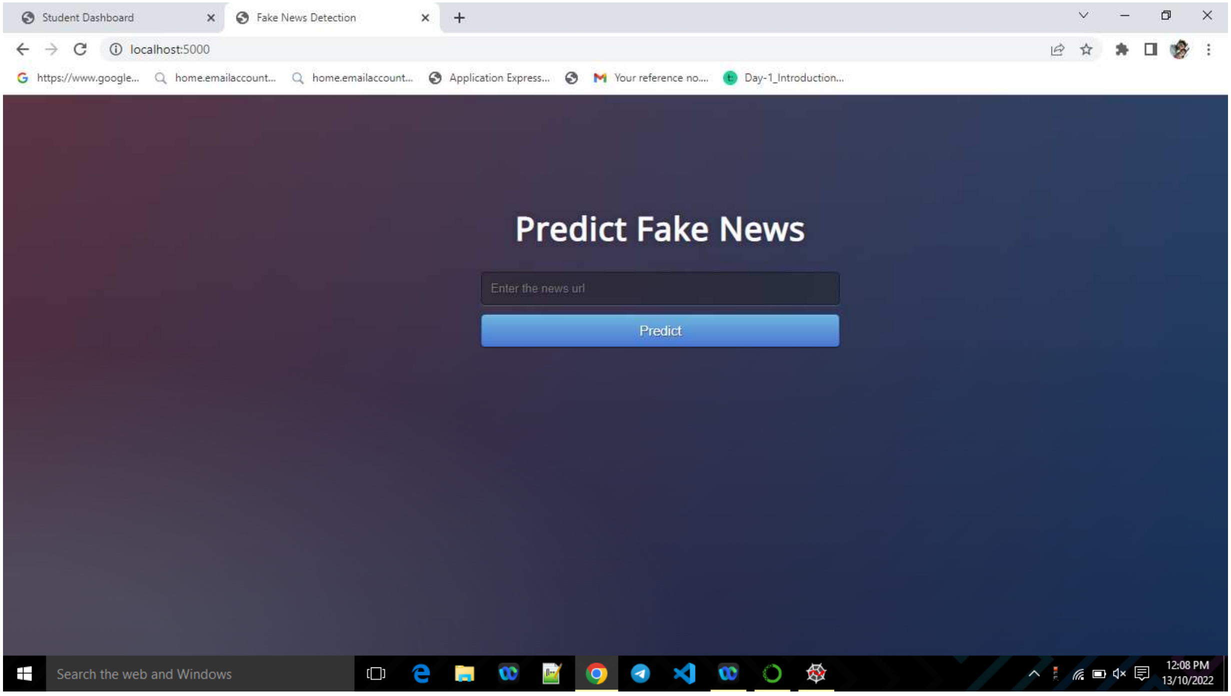
4. EXPERIMENTAL INVESTIGATIONS

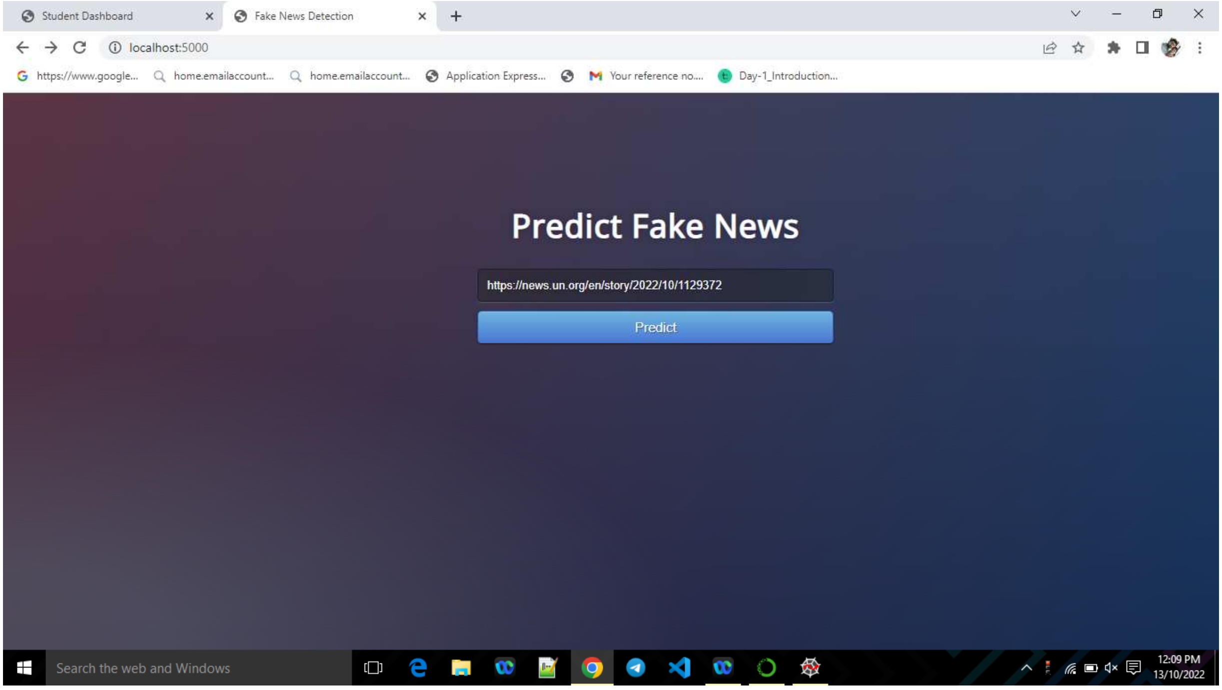
Study shows that how to find fakenews with we will passing any news url in the preposed system. Which will show the url passed news is real or fake

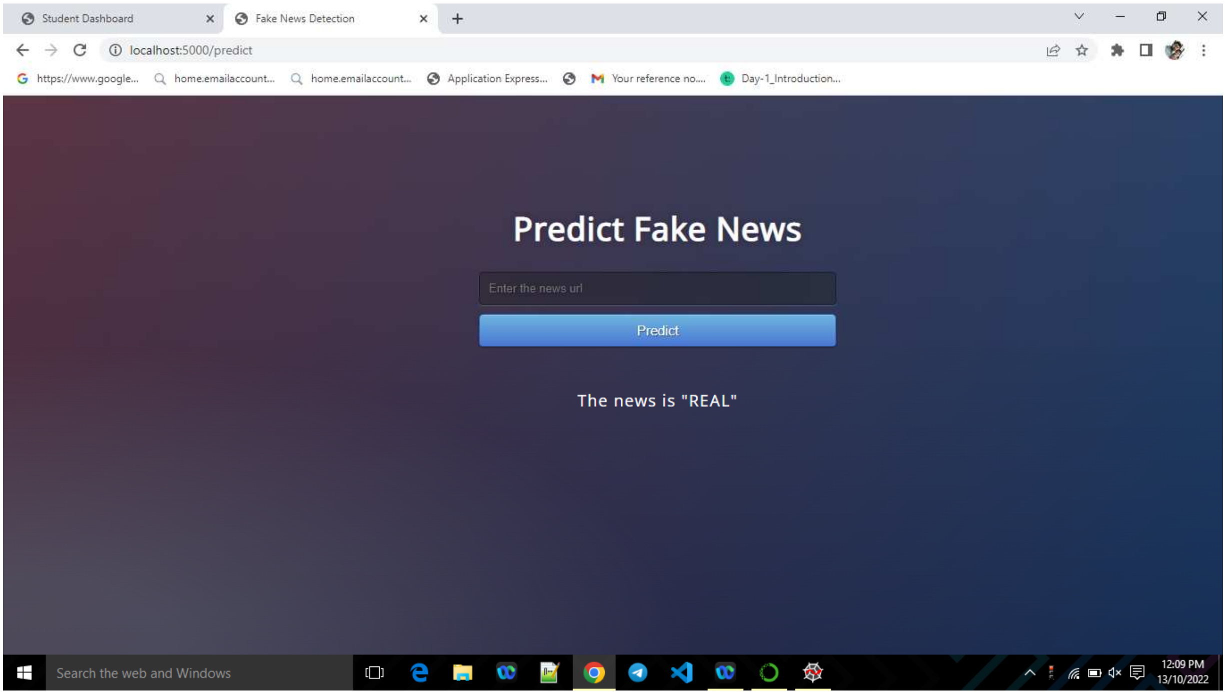
5. FLOWCHART

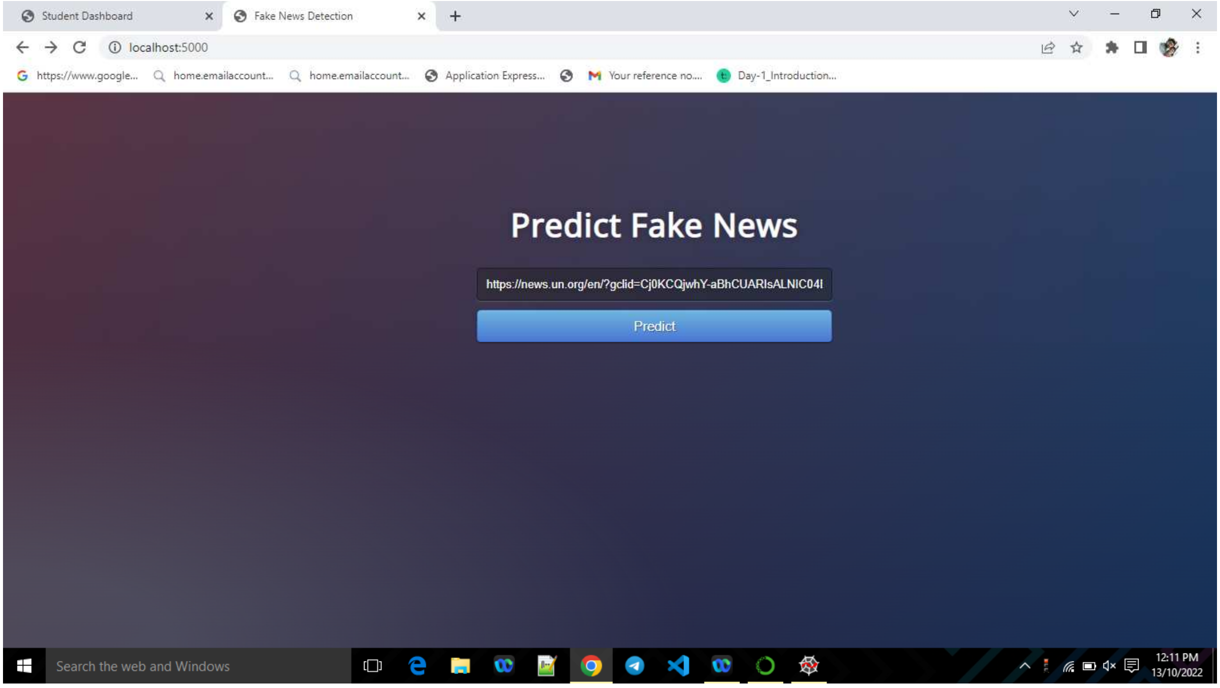

DATA COLLECTION
APPLICATION BUILDING
DATA PREPROCESSING
MODEL BUILDING

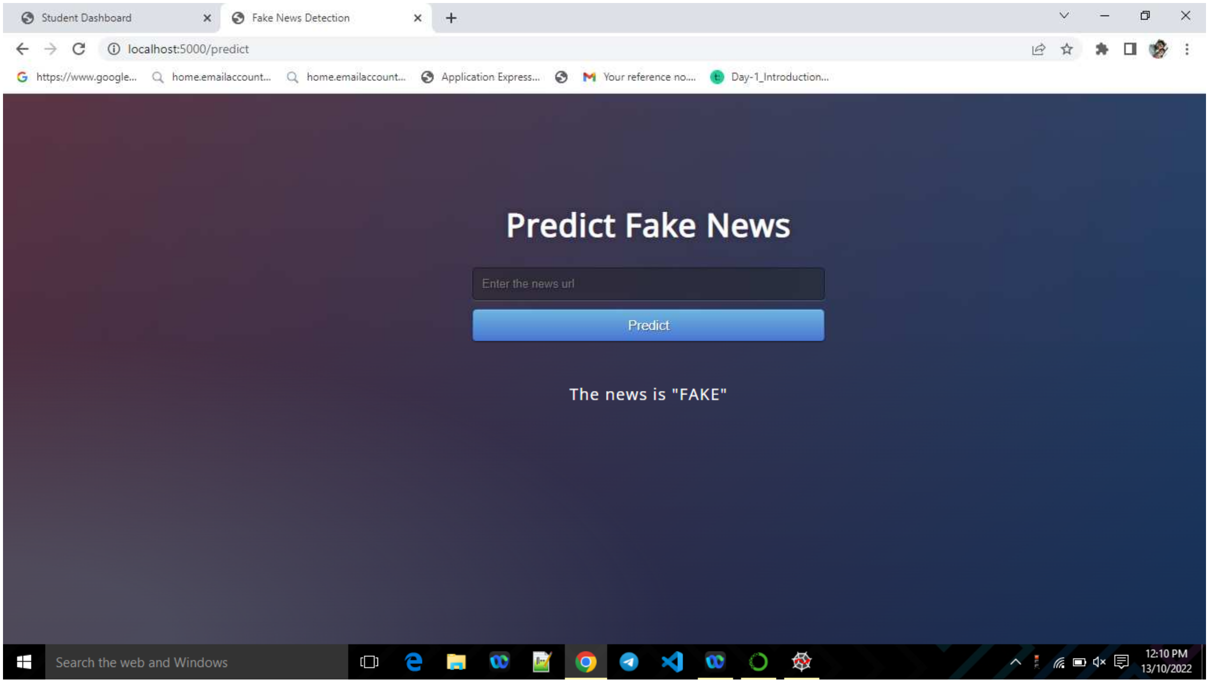
6. RESULT


Student Dashboard
Fake News Detection
localhost:5000
Predict Fake News
Enter the news url
Predict
Search the web and Windows
12:08 PM
13/10/2022


Student Dashboard
Fake News Detection
localhost:5000
Predict Fake News
https://news.un.org/en/story/2022/10/1129372
Predict
Search the web and Windows
12:09 PM
13/10/2022


Student Dashboard
Fake News Detection
localhost:5000/predict
Predict Fake News
Enter the news url
Predict
The news is "REAL"
Search the web and Windows
12:09 PM
13/10/2022





7. ADVANTAGES & DISADVANTAGES

Advantages:

The ubiquitous nature of social media platforms resulted into generation of large amount of multimedia data in social networks. The openness and unrestricted way to share the information on social media platforms fosters information spread across the network regardless of its credibility. Such kind of spreading the misinformation happens usually in the context of breaking news. Due to unverified information, such misinformation, also known as rumors may cause severe damages. Despite overwhelming use, uncontrolled nature of social media platforms usually results in generation and unfold of rumors. Therefore, automatically detecting the rumors from social media platforms is one of the highly sought-after research area in the domain of social media analytics. Motivated by the same, this paper focuses on detailed discussion of datasets and state-of-the-art approaches of rumor detection. Moreover, this paper sheds light upon supervised and unsupervised methods and deep learning approaches for rumor detection.

Disadvantages:

The passive aggressive model produces 93% accuracy. When we input the news text on the interface, it correctly identifies the news most of the time. We tested this by using news from The Onion. The Onion is a satire ‘news’ portal that posts fake funny news. When we pasted some of the news from the site on our web interface, those were correctly identified as fake. But when we wanted to test the news from BBC or New York Times, those were correctly identified as real. But the accuracy of the LSTM model was much lower, so we went with the Passive Aggressive model to produce output on the interface.

8. APPLICATIONS

- Deep Learning technology is used to detecting fake news in different socialmedia

9. CONCLUSION

In this project, we have established the application to predict the given news is fake or real based on the IBM cloud application. Fake news prediction can only use this web app to identify the real news

10. FUTURE SCOPE

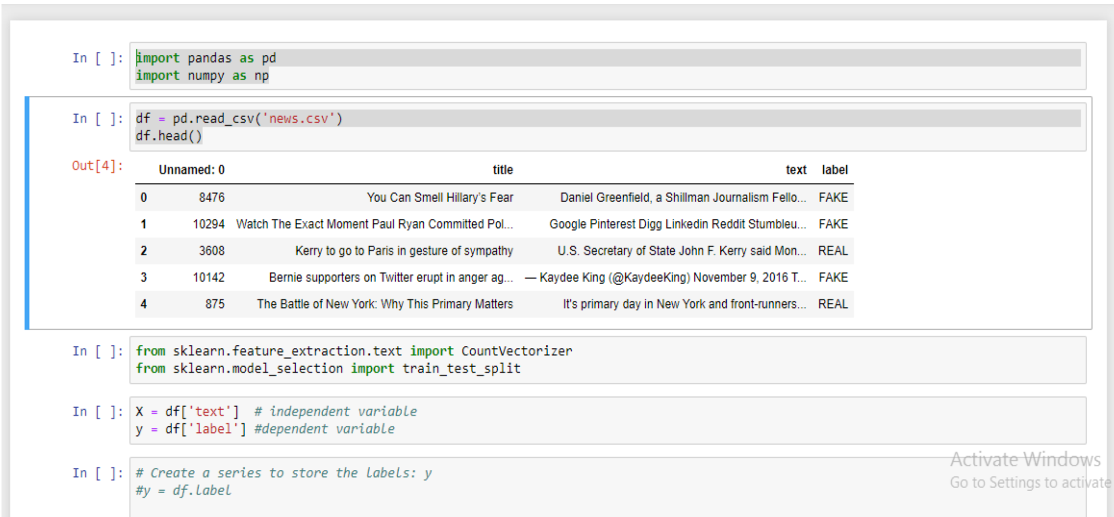
The project can be further enhanced by deploying the deep learning model obtained using a web application and larger dataset cloud be used for prediction to give higher accuracy and produce better result

11. BIBILOGRAPHY

- Linjie Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang, "A large-scale car dataset for fine-grained categorization and verification," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 3973–3981.
- Srimal Jayawardena et al., Image based automatic vehicle damage detection, Ph.D. thesis, Australian National University, 2013

APPENDIX

Source Code



```
In [ ]: import pandas as pd
        import numpy as np

In [ ]: df = pd.read_csv('news.csv')
        df.head()
```

Out[4]:

	Unnamed: 0	title	text	label
0	8476	You Can Smell Hillary's Fear	Daniel Greenfield, a Shillman Journalism Fello...	FAKE
1	10294	Watch The Exact Moment Paul Ryan Committed Pol...	Google Pinterest Digg Linkedin Reddit Stumbleu...	FAKE
2	3608	Kerry to go to Paris in gesture of sympathy	U.S. Secretary of State John F. Kerry said Mon...	REAL
3	10142	Bernie supporters on Twitter erupt in anger ag...	— Kaydee King (@KaydeeKing) November 9, 2016 T...	FAKE
4	875	The Battle of New York: Why This Primary Matters	It's primary day in New York and front-runners...	REAL

```
In [ ]: from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.model_selection import train_test_split

In [ ]: X = df['text']  # independent variable
        y = df['label'] #dependent variable

In [ ]: # Create a series to store the labels: y
        #y = df.label
```

```
In [ ]: # Create a series to store the labels: y
        #y = df.label

        # Create training set and test set
        X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                            test_size=0.33, random_state=53)

In [ ]: # Initialize a CountVectorizer object: count_vectorizer
        count_vectorizer = CountVectorizer(stop_words='english')

In [ ]: # Transform the training data using only the 'text' column values: count_train
        count_train = count_vectorizer.fit_transform(X_train)

        # Transform the test data using only the 'text' column values: count_test
        count_test = count_vectorizer.transform(X_test)

In [ ]: # Print the first 10 features of the count_vectorizer
        print(count_vectorizer.get_feature_names()[:10])

        ['00', '000', '0000', '00000031', '000035', '00006', '0001', '0001pt', '000ft', '000km']

In [ ]: from sklearn.feature_extraction.text import TfidfVectorizer

        # Initialize a TfidfVectorizer object: tfidf_vectorizer
        tfidf_vectorizer = TfidfVectorizer(stop_words='english', max_df=0.7)
```



```
In [ ]: from sklearn.feature_extraction.text import TfidfVectorizer

        # Initialize a TfidfVectorizer object: tfidf_vectorizer
        tfidf_vectorizer = TfidfVectorizer(stop_words='english', max_df=0.7)

        # Transform the training data: tfidf_train
        tfidf_train = tfidf_vectorizer.fit_transform(X_train)

        # transform the test data: tfidf_test
        tfidf_test = tfidf_vectorizer.transform(X_test)

        # Print the first 10 features
        print(tfidf_vectorizer.get_feature_names()[:10])

        # Print the first 5 vectors of the tfidf training data
        print(tfidf_train.A[:5])

        ['00', '000', '0000', '00000031', '000035', '00006', '0001', '0001pt', '000ft', '000km']
        [[0. 0. 0. ... 0. 0. 0.]
         [0. 0. 0. ... 0. 0. 0.]
         [0. 0. 0. ... 0. 0. 0.]
         [0. 0. 0. ... 0. 0. 0.]
         [0. 0. 0. ... 0. 0. 0.]]

In [ ]:
        count_df = pd.DataFrame(count_train.A, columns=count_vectorizer.get_feature_names())

        # Create the TfidfVectorizer DataFrame: tfidf_df
        tfidf_df = pd.DataFrame(tfidf_train.A, columns=tfidf_vectorizer.get_feature_names())
```

```
In [ ]:
count_df = pd.DataFrame(count_train.A, columns=count_vectorizer.get_feature_names())

# Create the TfidfVectorizer DataFrame: tfidf_df
tfidf_df = pd.DataFrame(tfidf_train.A, columns=tfidf_vectorizer.get_feature_names())

# Print the head of count_df
print(count_df.head())

# Print the head of tfidf_df
print(tfidf_df.head())

# Calculate the difference in columns: difference
difference = set(count_df.columns) - set(tfidf_df.columns)
print(difference)

# Check whether the DataFrame are equal
print(count_df.equals(tfidf_df))
```

```
   00  000  0000  00000031  000035  00006  0001  0001pt  000ft  000km  ...  \
0   0    0     0         0       0      0     0       0      0      0  ...
1   0    0     0         0       0      0     0       0      0      0  ...
2   0    0     0         0       0      0     0       0      0      0  ...
3   0    0     0         0       0      0     0       0      0      0  ...
4   0    0     0         0       0      0     0       0      0      0  ...

   والمرضى  هذا  من  محاولات  ما  لم  عن  عربي  حلب  ɐɞade
0    0     0   0   0   0       0   0   0      0      0
1    0     0   0   0   0       0   0   0      0      0
```



```
   والمرضى  هذا  من  محاولات  ما  لم  عن  عربي  حلب  ɐɞade
0    0     0   0   0   0       0   0   0      0      0
1    0     0   0   0   0       0   0   0      0      0
2    0     0   0   0   0       0   0   0      0      0
3    0     0   0   0   0       0   0   0      0      0
4    0     0   0   0   0       0   0   0      0      0

[5 rows x 56922 columns]
    00  000  0000  00000031  000035  00006  0001  0001pt  000ft  000km  ...  \
0  0.0  0.0   0.0       0.0     0.0    0.0   0.0     0.0    0.0    0.0  ...
1  0.0  0.0   0.0       0.0     0.0    0.0   0.0     0.0    0.0    0.0  ...
2  0.0  0.0   0.0       0.0     0.0    0.0   0.0     0.0    0.0    0.0  ...
3  0.0  0.0   0.0       0.0     0.0    0.0   0.0     0.0    0.0    0.0  ...
4  0.0  0.0   0.0       0.0     0.0    0.0   0.0     0.0    0.0    0.0  ...

   والمرضى  هذا  من  محاولات  ما  لم  عن  عربي  حلب  ɐɞade
0  0.0    0.0  0.0  0.0  0.0      0.0  0.0  0.0    0.0    0.0
1  0.0    0.0  0.0  0.0  0.0      0.0  0.0  0.0    0.0    0.0
2  0.0    0.0  0.0  0.0  0.0      0.0  0.0  0.0    0.0    0.0
3  0.0    0.0  0.0  0.0  0.0      0.0  0.0  0.0    0.0    0.0
4  0.0    0.0  0.0  0.0  0.0      0.0  0.0  0.0    0.0    0.0

[5 rows x 56922 columns]
set()
False
```

```
In [ ]:
from sklearn.naive_bayes import MultinomialNB

from sklearn.metrics import accuracy_score, confusion_matrix
```

```
: from sklearn.naive_bayes import MultinomialNB

  from sklearn.metrics import accuracy_score, confusion_matrix

  # Instantiate a Multinomial Naive Bayes classifier: nb_classifier
  nb_classifier = MultinomialNB()

  # Fit the classifier to the training data
  nb_classifier.fit(count_train, y_train)

  # Create the predicted tags: pred
  pred = nb_classifier.predict(count_test)

  # Calculate the accuracy score: score
  score = accuracy_score(y_test, pred)
  print(score)

  # Calculate the confusion matrix: cm
  cm =confusion_matrix(y_test, pred, labels=['FAKE', 'REAL'])
  print(cm)

  0.893352462936394
  [[ 865  143]
   [  80 1003]]

: count_test.shape

: (2091, 56922)
```



```
nb_classifier = MultinomialNB()

# Fit the classifier to the training data
nb_classifier.fit(tfidf_train, y_train)

# Create the predicted tags: pred
pred = nb_classifier.predict(tfidf_test)

# Calculate the accuracy score: score
score = accuracy_score(y_test, pred)
print(score)

# Calculate the confusion matrix: cm
cm = confusion_matrix(y_test, pred, labels=['FAKE', 'REAL'])
print(cm)

0.8565279770444764
[[ 739  269]
 [  31 1052]]

alphas = np.arange(0, 1, 0.1)

# Define train_and_predict()
def train_and_predict(alpha):
    # Instantiate the classifier: nb_classifier
    nb_classifier = MultinomialNB(alpha=alpha)

    # Fit to the training data
    nb_classifier.fit(tfidf_train, y_train)
```

```
]: alphas = np.arange(0, 1, 0.1)

# Define train_and_predict()
def train_and_predict(alpha):
    # Instantiate the classifier: nb_classifier
    nb_classifier = MultinomialNB(alpha=alpha)

    # Fit to the training data
    nb_classifier.fit(tfidf_train, y_train)

    # Predict the labels: pred
    pred = nb_classifier.predict(tfidf_test)

    # Compute accuracy: score
    score = accuracy_score(y_test, pred)
    return score

# Iterate over the alphas and print the corresponding score
for alpha in alphas:
    print('Alpha: ', alpha)
    print('Score: ', train_and_predict(alpha))
    print()
```

```
Alpha:  0.0
Score:  0.8813964610234337

Alpha:  0.1
Score:  0.8976566236250598
```



```
Score:  0.8976566236250598

Alpha:  0.2
Score:  0.8938307030129125

Alpha:  0.30000000000000004
Score:  0.8900047824007652

Alpha:  0.4
Score:  0.8857006217120995

Alpha:  0.5
Score:  0.8842659014825442

Alpha:  0.6000000000000001
Score:  0.874701099952176

Alpha:  0.7000000000000001
Score:  0.8703969392635102

Alpha:  0.8
Score:  0.8660927785748446

Alpha:  0.9
Score:  0.8589191774270684

C:\Users\HP\anaconda3\envs\tf1\lib\site-packages\sklearn\naive_bayes.py:508: UserWarning: alpha too small will result in numeri
c errors, setting alpha = 1.0e-10
  warnings.warn('alpha too small will result in numeric errors, '
```

```
C:\Users\HP\anaconda3\envs\tf1\lib\site-packages\sklearn\naive_bayes.py:508: UserWarning: alpha too small will result in numeri
c errors, setting alpha = 1.0e-10
  warnings.warn('alpha too small will result in numeric errors, '
```

```python
class_labels = nb_classifier.classes_

# Extract the features: feature_names
feature_names = tfidf_vectorizer.get_feature_names()

# Zip the feature names together with the coefficient array
# and sort by weights: feat_with_weights
feat_with_weights = sorted(zip(nb_classifier.coef_[0], feature_names))

# Print the first class label and the top 20 feat_with_weights entries
print(class_labels[0], feat_with_weights[:20])

# Print the second class label and the bottom 20 feat_with_weights entries
print(class_labels[1], feat_with_weights[-20:])
```

```
FAKE [(-11.316312804238807, '0000'), (-11.316312804238807, '000035'), (-11.316312804238807, '0001'), (-11.316312804238807, '000
1pt'), (-11.316312804238807, '000km'), (-11.316312804238807, '0011'), (-11.316312804238807, '006s'), (-11.316312804238807, '00
7'), (-11.316312804238807, '007s'), (-11.316312804238807, '008s'), (-11.316312804238807, '0099'), (-11.316312804238807, '00a
m'), (-11.316312804238807, '00p'), (-11.316312804238807, '00pm'), (-11.316312804238807, '014'), (-11.316312804238807, '015'),
(-11.316312804238807, '018'), (-11.316312804238807, '01am'), (-11.316312804238807, '020'), (-11.316312804238807, '023')]
REAL [(-7.742481952533027, 'states'), (-7.717550034444668, 'rubio'), (-7.703583809227384, 'voters'), (-7.654774992495461, 'hous
e'), (-7.649398936153309, 'republicans'), (-7.6246184189367, 'bush'), (-7.616556675728881, 'percent'), (-7.545789237823644, 'pe
ople'), (-7.516447881078008, 'new'), (-7.448027933291952, 'party'), (-7.411148410203476, 'cruz'), (-7.410910239085596, 'stat
e'), (-7.35748985914622, 'republican'), (-7.33649923948987, 'campaign'), (-7.2854057032685775, 'president'), (-7.21668781309177
55, 'sanders'), (-7.108263114902301, 'obama'), (-6.724771332488041, 'clinton'), (-6.5653954389926845, 'said'), (-6.328486029596
```

```python
feat_with_weights = sorted(zip(nb_classifier.coef_[0], feature_names))

# Print the first class label and the top 20 feat_with_weights entries
print(class_labels[0], feat_with_weights[:20])

# Print the second class label and the bottom 20 feat_with_weights entries
print(class_labels[1], feat_with_weights[-20:])
```

```
FAKE [(-11.316312804238807, '0000'), (-11.316312804238807, '000035'), (-11.316312804238807, '0001'), (-11.316312804238807, '000
1pt'), (-11.316312804238807, '000km'), (-11.316312804238807, '0011'), (-11.316312804238807, '006s'), (-11.316312804238807, '00
7'), (-11.316312804238807, '007s'), (-11.316312804238807, '008s'), (-11.316312804238807, '0099'), (-11.316312804238807, '00a
m'), (-11.316312804238807, '00p'), (-11.316312804238807, '00pm'), (-11.316312804238807, '014'), (-11.316312804238807, '015'),
(-11.316312804238807, '018'), (-11.316312804238807, '01am'), (-11.316312804238807, '020'), (-11.316312804238807, '023')]
REAL [(-7.742481952533027, 'states'), (-7.717550034444668, 'rubio'), (-7.703583809227384, 'voters'), (-7.654774992495461, 'hous
e'), (-7.649398936153309, 'republicans'), (-7.6246184189367, 'bush'), (-7.616556675728881, 'percent'), (-7.545789237823644, 'pe
ople'), (-7.516447881078008, 'new'), (-7.448027933291952, 'party'), (-7.411148410203476, 'cruz'), (-7.410910239085596, 'stat
e'), (-7.35748985914622, 'republican'), (-7.33649923948987, 'campaign'), (-7.2854057032685775, 'president'), (-7.21668781309177
55, 'sanders'), (-7.108263114902301, 'obama'), (-6.724771332488041, 'clinton'), (-6.5653954389926845, 'said'), (-6.328486029596
207, 'trump')]
```

```
C:\Users\HP\anaconda3\envs\tf1\lib\site-packages\sklearn\utils\deprecation.py:101: FutureWarning: Attribute coef_ was deprecate
d in version 0.24 and will be removed in 1.1 (renaming of 0.26).
  warnings.warn(msg, category=FutureWarning)
```

```python
import pickle
pickle.dump(nb_classifier,open('fake_news.pkl','wb'))
```