PHASE 3

Fake news detection using NLP

Introduction

Fake news is the intentional broadcasting of false or misleading claims as news, where the statements are purposely deceitful.

Newspapers, tabloids, and magazines have been supplanted by digital news platforms, blogs, social media feeds, and a plethora of mobile news applications. News organizations benefitted from the increased use of social media and mobile platforms by providing subscribers with up-to-the-minute information.

Consumers now have instant access to the latest news. These digital media platforms have increased in prominence due to their easy connectedness to the rest of the world and allow users to discuss and share ideas and debate topics such as democracy, education, health, research, and history. Fake news items on digital platforms are getting more popular and are used for profit, such as political and financial gain.

The problem

Because the Internet, social media, and digital platforms are widely used, anybody may propagate inaccurate and biased information. It is almost impossible to prevent the spread of fake news. There is a tremendous surge in the distribution of false news, which is not restricted to one sector such as politics but includes sports, health, history, entertainment, and science and research.

The Solution

It is vital to recognize and differentiate between false and accurate news. One method is to have an expert decide, and fact checks every piece of information, but this takes time and needs expertise that cannot be shared. Secondly, we can use machine learning and artificial intelligence tools to automate the identification of fake news.

Online news information includes various unstructured format data (such as documents, videos, and audio), but we will concentrate on text format news here. With the progress of

machine learning and Natural language processing, we can now recognize the misleading and false character of an article or statement.

Several studies and experiments are being conducted to detect fake news across all mediums.

Our main goal is:

- Explore and analyze the Fake News dataset.
- Build a classifier that can distinguish Fake news with as much accuracy as possible.

Implementation:

The following text preprocessing steps are performed here -

- Remove punctuations
- Convert text to tokens
- Remove tokens of length less than or equal to 3
- Remove stopwords using NLTK corpus stopwords list to match
- Apply lemmatization

```
# This Python 3 environment comes with many helpful analytics libraries installed # For example, here's several helpful packages to load

import numpy as np # linear algebra import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv) import itertools from sklearn.model_selection import train_test_split from sklearn.linear_model import PassiveAggressiveClassifier from sklearn.metrics import accuracy_score, confusion_matrix import string as st import re import nltk from nltk import PorterStemmer, WordNetLemmatizer import matplotlib.pyplot as plt

# Input data files are available in the read-only "../input/" directory # For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os for dirname, _, filenames in os.walk('/kaggle/input'): for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
# You can write up to 5GB to the current directory (/kaggle/working/) that gets
preserved as output when you create a version using "Save & Run All"  # You can
also write temporary files to /kaggle/temp/, but they won't be saved outside of
the current session


data = pd.read_csv('../input/textdb3/fake_or_real_news.csv') data.shape

data.head()

# Check how the labels are distributed print(np.unique(data['label']))
print(np.unique(data['label'].value_counts()))
```

Text cleaning and processing steps-

- Remove punctuations
- Convert text to tokens
- Remove tokens of length less than or equal to 3
- Remove stopwords using NLTK corpus stopwords list to match
- Apply lemmatization
- Convert words to feature vectors

```
# Remove all punctuations from the text

def remove_punct(text):
    return ("".join([ch for ch in text if ch not in st.punctuation]))

data['removed_punc'] = data['text'].apply(lambda x: remove_punct(x)) data.head()
''' Convert text to lower case tokens. Here, split() is applied on white-spaces.
But, it could be applied
    on special characters, tabs or any other string based on which text is to be
seperated into tokens.
''' def
tokenize(text):
    text = re.split('\s+' ,text)
return [x.lower() for x in text]

data['tokens'] = data['removed_punc'].apply(lambda msg : tokenize(msg)) data.head()

# Remove tokens of length less than 3 def
remove_small_words(text):     return [x for x
in text if len(x) > 3 ]
```

```
data['filtered_tokens']  =
data['tokens'].apply(lambda    x     :
remove_small_words(x)) data.head()

''' Remove stopwords. Here, NLTK corpus list is used for a match. However, a
customized user-defined     list could be created and used to limit the
matches in input text.
''' def
remove_stopwords(text):
    return [word for word in text if word not in
nltk.corpus.stopwords.words('english')]  data['clean_tokens'] =
data['filtered_tokens'].apply(lambda x :
remove_stopwords(x)) data.head()

# Apply lemmatization on tokens def lemmatize(text):    word_net
= WordNetLemmatizer()     return [word_net.lemmatize(word) for
word in text]

data['lemma_words'] = data['clean_tokens'].apply(lambda x : lemmatize(x))
data.head()

# Create sentences to get clean text as input for vectors

def return_sentences(tokens):    return "
".join([word for word in tokens])

data['clean_text'] = data['lemma_words'].apply(lambda x : return_sentences(x))
data.head()

# Generate a basic word cloud from wordcloud import
WordCloud, ImageColorGenerator

text = " ".join([x for x in data['clean_text']]) # Create  and
generate a word cloud image:
wordcloud = WordCloud(max_font_size=30, max_words=1000).generate(text)

# Display the generated image: plt.figure(figsize= [20,10])
plt.imshow(wordcloud) plt.axis("off") plt.show()

# Prepare data for the model. Convert label in to binary

data['label'] = [1 if x == 'FAKE' else 0 for x in data['label']] data.head()

# Split the dataset

X_train,X_test,y_train,y_test = train_test_split(data['clean_text'],
data['label'], test_size=0.2, random_state = 5)
```

```
print(X_train.shape) print(X_test.shape)
```

TF-IDF : Term Frequency - Inverse Document Frequency

The term frequency is the number of times a term occurs in a document. Inverse document frequency is an inverse function of the number of documents in which that a given word occurs.

The product of these two terms gives tf-idf weight for a word in the corpus. The higher the frequency of occurrence of a word, lower is it's weight and vice-versa. This gives more weightage to rare terms in the corpus and penalizes more commonly occuring terms.

Other widely used vectorizer is Count vectorizer which only considers the frequency of occurrence of a word across the corpus.

```
                        from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer() tfidf_train =  tfidf.fit_transform(X_train)
tfidf_test = tfidf.transform(X_test)

print(tfidf_train.toarray())
print(tfidf_train.shape) print(tfidf_test.toarray())
print(tfidf_test.shape)
```

Passive Aggressive Classifiers

- Passive Aggressive algorithms are online learning algorithms. Such an algorithm remains passive for a correct classification outcome, and turns aggressive in the event of a miscalculation, updating and adjusting.
- Passive: If the prediction is correct, keep the model and do not make any changes. i.e., the data in the example is not enough to cause any changes in the model.
- Aggressive: If the prediction is incorrect, make changes to the model. i.e., some change to the model may correct it.

These are typically used for large datasets where batch learning is not possible due to huge volumes of frequently incoming data.

- Some important parameters -
- C : This is the regularization parameter, and denotes the penalization the model will make on an incorrect prediction
- max_iter : The maximum number of iterations the model makes over the training data.
- tol : The stopping criterion.

We are using a simple implementation of this model here.

```
# Passive Aggresive Classifier pac =
PassiveAggressiveClassifier(max_iter=50) pac.fit(tfidf_train,y_train) pred
=   pac.predict(tfidf_test) print("Accuracy  score  :
{}".format(accuracy_score(y_test, pred))) print("Confusion matrix : \n
{}".format(confusion_matrix(y_test, pred)))

# Logistic Regression model from
sklearn.linear_model import LogisticRegression

lr = LogisticRegression(max_iter = 500) lr.fit(tfidf_train,
y_train) print('Logistic
Regression model fitted..')

pred   =   lr.predict(tfidf_test)  print("Accuracy   score   :
{}".format(accuracy_score(y_test, pred))) print("Confusion matrix : \n
{}".format(confusion_matrix(y_test, pred)))
```

Logistic Regression could not outperform XGBoost and LGBM but its performance is considerably close to them and it is much less complex.

```
                                        import xgboost from xgboost import XGBClassifier

xgb = XGBClassifier() xgb.fit(tfidf_train,  y_train)

print('XGBoost  Classifier  model  fitted..')  pred  =  xgb.predict(tfidf_test)
print("Accuracy score : {}".format(accuracy_score(y_test, pred))) print("Confusion
matrix  :  \n  {}".format(confusion_matrix(y_test,  pred)))    import  lightgbm  from
lightgbm import LGBMClassifier

lgbm = LGBMClassifier() lgbm.fit(tfidf_train,  y_train)

print('LightGBM  Classifier  model  fitted..') pred  =  lgbm.predict(tfidf_test)
print("Accuracy  score  :
{}".format(accuracy_score(y_test, pred))) print("Confusion matrix : \n
{}".format(confusion_matrix(y_test, pred)))
```

OUTPUT:

['FAKE' 'REAL']

[3164 3171]

(5068,)

(1267,) [[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.]

[0. 0. 0. ... 0. 0. 0.]]

(5068, 68134)

[[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.]

[0. 0. 0. ... 0. 0. 0.]]

(1267, 68134)

Accuracy score : 0.9329123914759274 Confusion

matrix :

[[588 42]

[43 594]]

Logistic Regression model fitted..

Accuracy score : 0.9179163378058406

Confusion matrix :

[[565 65]

[39 598]]

XGBoost Classifier model fitted..

Accuracy score : 0.9273875295974744

Confusion matrix :

[[578 52]

[40 597]]

[LightGBM] [Info] Number of positive: 2527, number of negative: 2541

[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.348799 seconds.

You can set `force_row_wise=true` to remove the overhead.

And if memory is not enough, you can set `force_col_wise=true`.

[LightGBM] [Info] Total Bins 354681

[LightGBM] [Info] Number of data points in the train set: 5068, number of used features: 8289

[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.498619 -> initscore=-0.005525

[LightGBM] [Info] Start training from score -0.005525

LightGBM Classifier model fitted.. Accuracy score :

0.9321231254932912 Confusion matrix :

[[584 46]

[40 597]]

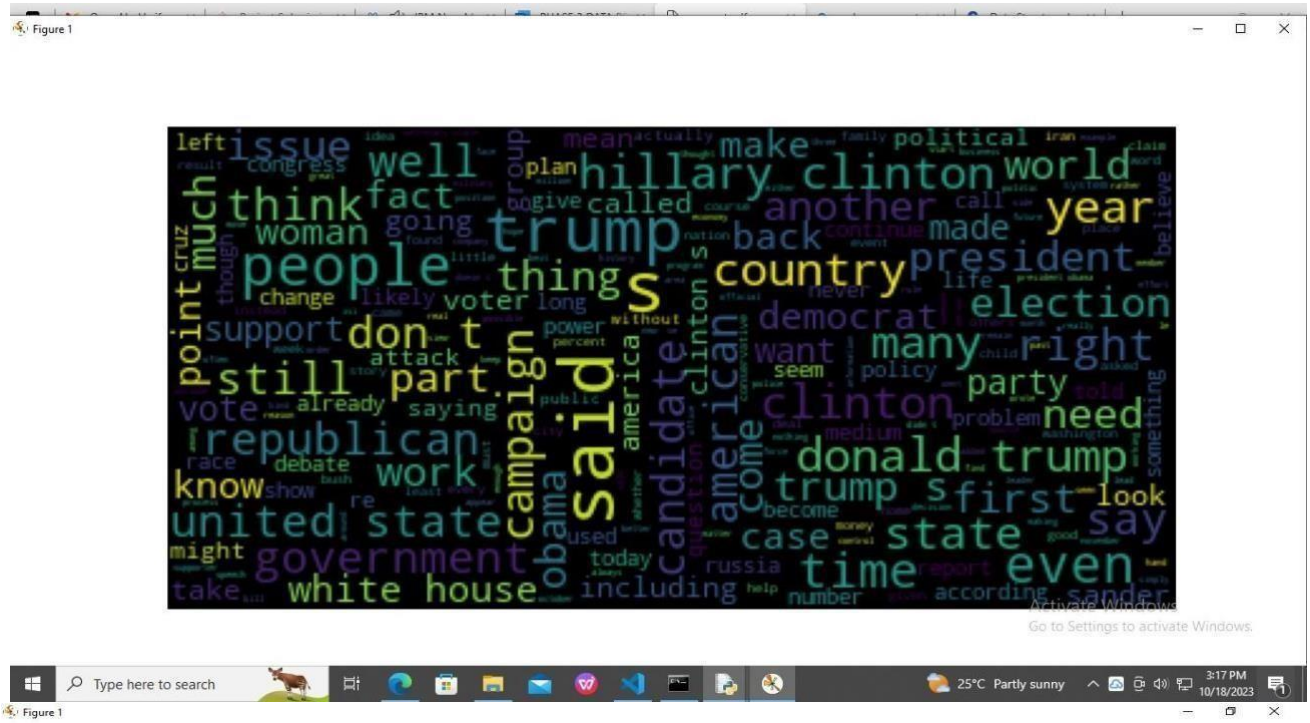
Figure 1
issue
well
hillary
clinton
world
think
fact
trump
another
year
woman
people
thing
country
president
election
support
don t
democrat
many
right
still
part
said
campaign
candidate
american
clinton
party
need
republican
donald
trump
know
work
united
state
first
say
government
state
time
even
white house
including
Activate Windows
Go to Settings to activate Windows.
Type here to search
25°C Partly sunny
3:17 PM
10/18/2023

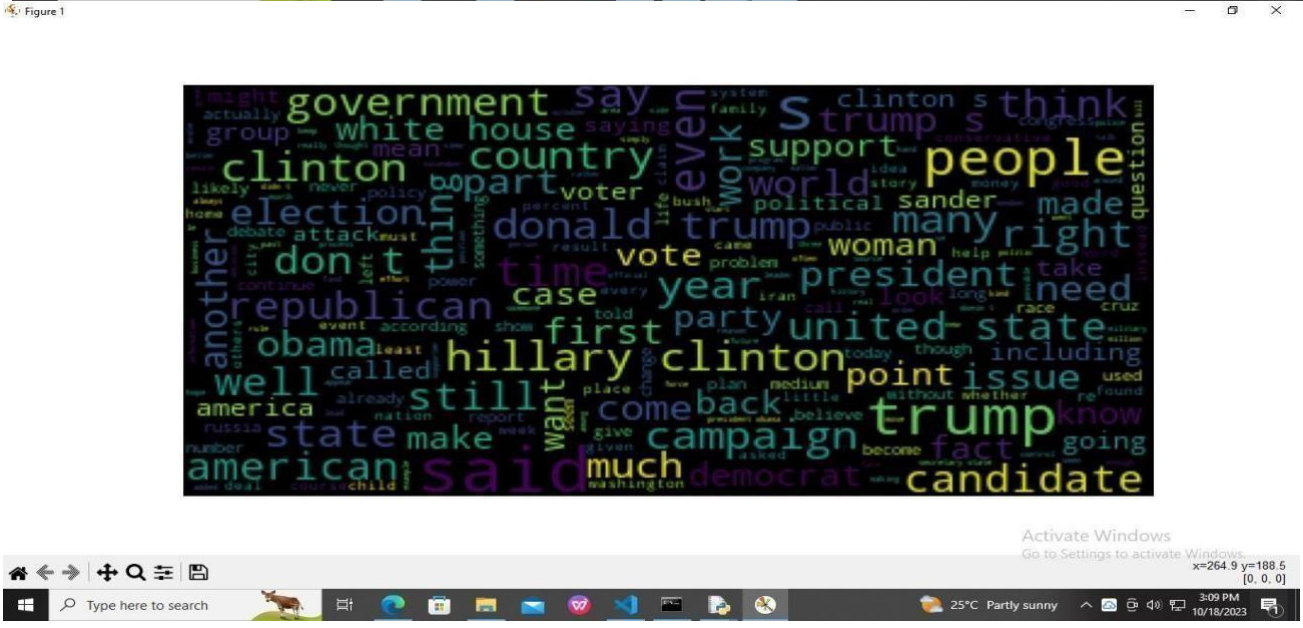
Figure 1
government
say
clinton
think
white house
trump
support
people
clinton
country
world
election
donald
trump
many
right
don t
time
vote
woman
president
need
republican
case
year
first
party
united
state
obama
hillary
clinton
point
issue
well
still
come
back
trump
state
make
campaign
american
said
much
democrat
candidate
Activate Windows
Go to Settings to activate Windows.
x=264.9 y=188.5
[0, 0, 0]
Type here to search
25°C Partly sunny
3:09 PM
10/18/2023