National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Shaik Nasir Vali
Student ID: 21166421

School of Computing
National College of Ireland

Supervisor: Hicham Rifai

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Shaik Nasir Vali		
Student ID:	21166421		
Programme :	Data Analytics	**Year:**	2022
Module:	MSc Research Project		
Lecturer:	Hicham Rifai		
Submission Due Date:	15 / 12 / 2022		
Project Title:	Fake news detection using Deep learning and NLP		
Word Count:	715	**Page Count:**	14

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Shaik Nasir Vali
Date:	15/12/2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Fake News detection using Deep learning and Natural Language Processing

Shaik Nasir Vali
Student ID: 21166421

1 Introduction

The configuration manual illustrates how to execute the research topic "Fake news detection using Deep Learning and NLP" step by step. The next sections have the details about software and hardware requirements for implementation. In order, the required computer code, associated aim, and output results are displayed. Machine Learning algorithms such as Decision tree classifier, Logistic Regression, NLP techniques and RNN with LSTM were used.

2 System Requirements

This section describes the system requirements for successfully implementing the project, and prior knowledge of the system specification is always before conducting experiments.

2.1 Hardware

The specifications utilized on the local system are as follows:

1. Hard-Disk Memory - 500GB (SDD)
2. Processor – Intel i5-6200U CPU
3. RAM – 8GB
4. System OS – 64-bit Windows 11

2.2 Software

1. Python 3.9 – Python covers most of the project
2. Microsoft Excel – CSV file format was used through Excel
3. Postgres SQL – It is used to load the data in jupyter notebook
4. Jupyer Notebook – It is an IDE platform which can be accessed online, this was used to write python code from beginning till end.

3 Data Pre-processing and Evaluation

3.1 Installing packages

There are many packages that were installed to carry on particular tasks. The following figures show all the packages that were imported :

```
import pandas as p
import matplotlib.pyplot as plt
import seaborn as s
import numpy as n
```

```
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
import re
import string
```

```
from wordcloud import WordCloud
```

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.tree import DecisionTreeClassifier
```

```
from tensorflow.keras.models import Sequential,load_model
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.layers import Dropout, Activation, Flatten
import warnings
warnings.filterwarnings("ignore")
```

```
import tqdm
import tensorflow as tf
import numpy as np
import keras_metrics # for recall and precision metrics
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
from keras.layers import Embedding, LSTM, Dropout, Dense
from keras.models import Sequential
from keras.utils import to_categorical
from keras.callbacks import ModelCheckpoint, TensorBoard
from sklearn.model_selection import train_test_split
import time
import numpy as np
import pickle
from keras.preprocessing.text import Tokenizer
```

Figure 1: Installing packages

3.2 Data cleaning

The data which was taken from the public open-source site was already clean without any null values or duplicate values.

```
df.shape
```

```
(3988, 4)
```

```
#show columns
df.columns
```

```
Index(['URLs', 'Headline', 'Body', 'Label'], dtype='object')
```

```
#To describe the dataframe
df.describe()
```

	Label
count	3988.000000
mean	0.468405
std	0.499063
min	0.000000
25%	0.000000
50%	0.000000
75%	1.000000
max	1.000000

```
#Checking datatype and information about dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3988 entries, 0 to 4008
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   URLs      3988 non-null   object
 1   Headline  3988 non-null   object
 2   Body      3988 non-null   object
 3   Label     3988 non-null   int64 
dtypes: int64(1), object(3)
memory usage: 155.8+ KB
```

Checking duplicate values of dataframe:

```
#Checking for duplicate data
df.duplicated()
```

```
0       False
1       False
2       False
3       False
4       False
        ...
4003    False
4004    False
4005    False
4007    False
4008    False
Length: 3988, dtype: bool
```

```
#find sum of duplicate data
sum(df.duplicated())
```

```
0
```

```
#Checking sum of missing values
df.isnull().sum()
```

```
URLs        0
Headline    0
Body        0
Label       0
dtype: int64
```

Figure 2: Data cleaning

3.3 Encoding

Label Encoder was imported from sklearn and used in this study.

```
from sklearn.preprocessing import LabelEncoder
var_mod = ['URLs', 'Headline', 'Body', 'Label']
le = LabelEncoder()
for i in var_mod:
    df[i] = le.fit_transform(df[i]).astype(str)
```

```
df.head()
```

	URLs	Headline	Body	Label
0	1617	880	1288	1
1	2933	1395	1521	1
2	2598	2471	2431	1
3	3071	745	1580	1
4	2060	1237	863	1

Figure 3: Data Encoding

3.4 Stemming, Normalization and Removal of Whitespaces & Stopwords

```
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
import re
import string
# remove whitespaces
df['Headline']=df['Headline'].str.strip()
# Lowercase the text
df['Headline'] = df['Headline'].str.lower()
#remove punctuation
punc = string.punctuation
table = str.maketrans('','',punc)
df['Headline']=df['Headline'].apply(lambda x: x.translate(table))
# tokenizing each message
df['word_tokens']=df.apply(lambda x: x['Headline'].split(' '),axis=1)
# removing stopwords
df['cleaned_text'] = df.apply(lambda x: [word for word in x['word_tokens'] if word not in stopwords.words('english')],axis=1)
# stemming
ps = PorterStemmer()
df['stemmed']= df.apply(lambda x: [ps.stem(word) for word in x['cleaned_text']],axis=1)
# remove single letter words
df['final_text'] = df.apply(lambda x: ' '.join([word for word in x['stemmed'] if len(word)>1]),axis=1)
```

Figure 4: Stemming, Normalization and Removal of whitespaces & stopwords

3.5 Evaluation of Models

3.5.1 Decision Tree Classifier

Decision Tree Classifier :

```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()

dtree.fit(X_train,y_train)
predictR = dtree.predict(X_test)
print("")
x = (accuracy_score(y_test,predictR)*100)
print('Accuracy result ofDecision Tree Classifier  is:', x)
print("")

print("")
print('Classification report of Decision Tree Classifier : Results:')
print("")

print(classification_report(y_test,predictR))
xd = (accuracy_score(y_test,predictR)*100)


cm2=confusion_matrix(y_test,predictR)
print('Confusion Matrix result of Decision Tree Classifier : is:\n', confusion_matrix(y_test,predictR))
print("")

sensitivity1 = cm2[0,0]/(cm2[0,0]+cm2[0,1])
print('Sensitivity : ', sensitivity1 )
print("")
specificity1 = cm2[1,1]/(cm2[1,0]+cm2[1,1])
print('Specificity : ', specificity1)
```

Figure 5: Decision Tree Classifier

3.5.2 Logistic Regression

Logistic Regression:

```
from sklearn.linear_model import LogisticRegression
logR= LogisticRegression()
logR.fit(X_train,y_train)
predictR = logR.predict(X_test)
print("")
x = (accuracy_score(y_test,predictR)*100)
print('Accuracy result of Logistic Regression is:', x)
print("")

print("")
print('Classification report of Logistic Regression : Results:')
print("")

print(classification_report(y_test,predictR))
xl = (accuracy_score(y_test,predictR)*100)


cm2=confusion_matrix(y_test,predictR)
print('Confusion Matrix result of Logistic Regression : is:\n', confusion_matrix(y_test,predictR))
print("")

sensitivity1 = cm2[0,0]/(cm2[0,0]+cm2[0,1])
print('Sensitivity : ', sensitivity1 )
print("")
specificity1 = cm2[1,1]/(cm2[1,0]+cm2[1,1])
print('Specificity : ', specificity1)
```

Figure 6: Logistics Regression

3.5.3 Natural Language Toolkit

Figure 2: Data cleaning

```
# divide the set in training and test
from sklearn.model_selection import train_test_split
X,X_test,y,y_test = train_test_split(df.loc[:,'Headline':],df['Label'],test_size=0.2)
```

```
# Now we'll create a vocabulary for the training set with word count
from collections import defaultdict
vocab=defaultdict(int)
for text in X['final_text'].values:
    for elem in text.split(' '):
        vocab[elem]+=1
```

```
#!pip install wordcloud
```

```
from wordcloud import WordCloud
# Now we look at the types of words in ham and spam. We plot wordclouds for both
ham_text=' '.join(X.loc[y==0,'final_text'].values)
ham_wordcloud = WordCloud(background_color='white',max_words=2000).generate(ham_text)
spam_text=' '.join(X.loc[y==1,'final_text'].values)
spam_wordcloud = WordCloud(background_color='white',max_words=2000).generate(spam_text)
plt.figure(figsize=[20,30])
plt.subplot(1,2,1)
plt.imshow(spam_wordcloud,interpolation='bilinear')
plt.title('news:fake')
plt.axis('off')
plt.subplot(1,2,2)
plt.imshow(ham_wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('news:real')
```

Figure 7: Natural Language Toolkit

3.5.4 RNN with LSTM

```
from sklearn.metrics import confusion_matrix, classification_report, matthews_corrcoef, cohen_kappa_score, accuracy_score, averag
```

```
X = data.drop(labels='Label', axis=1)
#Response variable
y = data.loc[:,'Label']
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=1, stratify=y)
```

```
scaler = MinMaxScaler()
data_training = scaler.fit_transform(X_train)
data_training
```

```
array([[0.57717718, 0.41849752, 0.5027972 ],
       [0.77297297, 0.56803685, 0.66573427],
       [0.74024024, 0.71545004, 0.81923077],
       ...,
       [0.17597598, 0.0368533 , 0.08146853],
       [0.07207207, 0.03437279, 0.72202797],
       [0.82342342, 0.51169383, 0.99125874]])
```

```
scaler = MinMaxScaler()
data_testing = scaler.fit_transform(X_test)
data_testing
```

```
array([[0.67257435, 0.92416726, 0.32156589],
       [0.3986182 , 0.50815025, 0.56623558],
       [0.70501652, 0.88837704, 0.19888151],
       ...,
       [0.11745269, 0.4280652 , 0.49912618],
       [0.34094323, 0.52019844, 0.09751835],
       [0.28356864, 0.54854713, 0.60887801]])
```

Figure 8: RNN with LSTM

```
data_training[0:10]
```

```
array([[0.57717718, 0.41849752, 0.5027972 ],
       [0.77297297, 0.56803685, 0.66573427],
       [0.74024024, 0.71545004, 0.81923077],
       [0.41231231, 0.97306875, 0.96888112],
       [0.53363363, 0.71899362, 0.52307692],
       [0.64204204, 0.51098512, 0.32132867],
       [0.51321321, 0.04429483, 0.80594406],
       [0.66126126, 0.52232459, 0.8513986 ],
       [0.56186186, 0.95038979, 0.03951049],
       [0.7039039 , 0.90077959, 0.98461538]])
```

```
data_testing[0:10]
```

```
array([[0.67257435, 0.92416726, 0.32156589],
       [0.3986182 , 0.50815025, 0.56623558],
       [0.70501652, 0.88837704, 0.19888151],
       [0.63772905, 0.37632884, 0.41873471],
       [0.38329829, 0.53437279, 0.51031108],
       [0.13848002, 0.66158753, 0.92205523],
       [0.69990988, 0.70056697, 0.23383432],
       [0.03574647, 0.64989369, 0.70954212],
       [0.6581556 , 0.32813607, 0.86158686],
       [0.8227696 , 0.34691708, 0.8409647 ]])
```

```
X_train1 = []
y_train1 = []
```

```
for i in range(60, data_training.shape[0]):
    X_train1.append(data_training[i-60:i])
    y_train1.append(data_training[i, 0])
```

```
X_train, y_train = np.array(X_train1), np.array(y_train1)
```

```
X_train.shape
```

```
(1934, 60, 3)
```

Figure 9: Data training

Figure 10:Data training for RNN

```
scores = model.predict(X_test)
```

```
import math, time
```

```
print("")
trainScore = model.evaluate(X_train, y_train, verbose=0)
print('Train Score: %.2f MSE (%.2f RMSE)' % (trainScore[0], math.sqrt(trainScore[0])))
print("")
testScore = model.evaluate(X_test, y_test, verbose=0)
print('Test Score: %.2f MSE (%.2f RMSE)' % (testScore[0], math.sqrt(testScore[0])))
```

```
# Initialising the RNN
model = Sequential()
# Adding the first LSTM layer and some Dropout regularisation
model.add(LSTM(24, return_sequences=True,input_shape=(X_train.shape[1], X_train.shape[2])))  # returns a sequence of vectors of
model.add(Dropout(0.2))

# Adding a second LSTM layer and some Dropout regularisation
model.add(LSTM(units = 50))
#model.add(Dropout(0.2))

model.add(Dense(10,activation='relu'))


# Adding the output layer
#model.add(Dense(1, activation="linear"))
model.add(Dense(1))

#model.add(Dense(units = 1))
# Compiling the RNN

model.compile(optimizer = 'adam', loss = 'mean_squared_error',metrics=['accuracy'])

history=model.fit(X_train, y_train,batch_size=2, epochs=10)
# Model summary for number of parameters use in the algorithm
model.summary()
```

```
Train on 3868 samples
Epoch 1/10
3868/3868 [==============================] - 59s 15ms/sample - loss: 0.0860 - accuracy: 2.5853e-04
Epoch 2/10
3868/3868 [==============================] - 55s 14ms/sample - loss: 0.0827 - accuracy: 5.1706e-04
Epoch 3/10
3868/3868 [==============================] - 55s 14ms/sample - loss: 0.0825 - accuracy: 5.1706e-04
Epoch 4/10
3868/3868 [==============================] - 55s 14ms/sample - loss: 0.0823 - accuracy: 5.1706e-04
Epoch 5/10
3868/3868 [==============================] - 55s 14ms/sample - loss: 0.0823 - accuracy: 2.5853e-04
Epoch 6/10
3868/3868 [==============================] - 55s 14ms/sample - loss: 0.0820 - accuracy: 2.5853e-04
Epoch 7/10
3868/3868 [==============================] - 55s 14ms/sample - loss: 0.0819 - accuracy: 5.1706e-04
Epoch 8/10
3868/3868 [==============================] - 56s 14ms/sample - loss: 0.0820 - accuracy: 5.1706e-04
Epoch 9/10
3868/3868 [==============================] - 55s 14ms/sample - loss: 0.0817 - accuracy: 2.5853e-04
Epoch 10/10
3868/3868 [==============================] - 55s 14ms/sample - loss: 0.0816 - accuracy: 5.1706e-04
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 60, 24)            2688
_________________________________________________________________
dropout (Dropout)            (None, 60, 24)            0
_________________________________________________________________
lstm_1 (LSTM)                (None, 50)                15000
_________________________________________________________________
dense (Dense)                (None, 10)                510
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 11
=================================================================
Total params: 18,209
Trainable params: 18,209
Non-trainable params: 0
_________________________________________________________________
```

3.5.5 RNN with LSTM using GloVe embedding vectors

```
SEQUENCE_LENGTH = 100 # the length of all sequences (number of words per sample)
EMBEDDING_SIZE = 100  # Using 100-Dimensional GloVe embedding vectors
TEST_SIZE = 0.20 # ratio of testing set

BATCH_SIZE = 64
EPOCHS =30 # number of epochs

# to convert labels to integers and vice-versa
#label2int = {"truthful": 0, "deceptive": 1} #added
#int2label = {0: "truthful", 1: "deceptive"} #added

label2int = {'Real':1, 'Fake':0} #added
int2label = {0:'Fake', 1:'Real'} #added
```

```
x=df['Headline']
```

```
x
```

```
0              Four ways Bob Corker skewered Donald Trump
1       Linklater's war veteran comedy speaks to moder...
2       Trump's Fight With Corker Jeopardizes His Legi...
3       Egypt's Cheiron wins tie-up with Pemex for Mex...
4             Jason Aldean opens 'SNL' with Vegas tribute
                              ...
4003    CNN and Globalist Exposed - Steve Quayle and A...
4004                                    Trends to Watch
4005    Trump Jr. Is Soon To Give A 30-Minute Speech F...
4007    China to accept overseas trial data in bid to ...
4008    Vice President Mike Pence Leaves NFL Game Beca...
Name: Headline, Length: 3988, dtype: object
```

```
x=list(x)
```

```
x
```

```
'CNN Hero Mona Patel: Sparkle Twins',
'Las Vegas Shooting Makes Country Guitarist Change His Mind On Guns',
'Netflix fends off criticism over Canada investment',
'Huge Triangle Shadow In The Sky In Louisiana',
'Kenyan Election Officials Open New Vote to All Original Candidates',
'Some Tips for Selecting The Best Flower Bouquet Online',
'12 Rohingya refugees killed after boat capsizes between Myanmar and Bangladesh',
'Nelly arrested on suspicion of rape',
'Happy 350,000th birthday: Study pushes back Homo sapiens origins',
'Kids & Fitness....Get the Kids Off the Couch & Have Some Fun!',
"Exclusive: EU regulators set to approve Atlantia's bid for Abertis - sources",
'France still not confirmed IS links to Marseille attacker',
"North Korea's Other Threat: Why EMP Becomes An Extinction Level Event",
'Where Is President Trump's Sense Of Compassion?',
'New Documents Show Pentagon Is Shipping Billions Worth Of Weapons To Terrorists In Syria, Trying To Cover Its Tracks',
'Jimmy Kimmel Weeps As He Talks About Shooting In His Hometown',
'Wonder Woman's Surprising Back Story Has a Film of Its Own',
'Mom Fights Back Home Invaders On Detroit's West Side',
'Gatekeeper Invades The 312 Podcastto Preview the Blackhawks Season',
"'Dancing With the Stars' Las Vegas tribute",
```

```
df.Headline.unique()
```

```
array(['Four ways Bob Corker skewered Donald Trump',
       "Linklater's war veteran comedy speaks to modern America, says star",
       'Trump's Fight With Corker Jeopardizes His Legislative Agenda',
       ...,
       'Hervé Leroux, Creator of the Hervé Léger Bandage Dress, Dies at 60',
       'Trump Jr. Is Soon To Give A 30-Minute Speech For $100,000',
       'Vice President Mike Pence Leaves NFL Game Because of Anti-American Protests'],
      dtype=object)
```

```
df['Label'] = df['Label'].replace({0:'Fake', 1:'Real'})
```

```
y=df['Label']
```

```
y.unique()
```

```
array(['Real', 'Fake'], dtype=object)
```

```
y=list(y)
```

```
y
```

```
'Fake',
'Fake',
'Fake',
'Fake',
'Real',
'Real',
'Real',
'Real',
'Fake',
'Real',
'Fake',
'Real',
'Fake',
'Real',
'Fake',
'Real',
'Real',
'Real',
'Fake',
'Real',
```

Figure 11: RNN with LSTM using GloVe embedding vectors

```
# Text tokenization
# vectorizing text, turning each text into sequence of integers

tokenizer = Tokenizer(num_words=None,char_level=True,oov_token='UNK')
tokenizer.fit_on_texts(x)
```

```
# convert to sequence of integers
x = tokenizer.texts_to_sequences(x)
```

```
print(x[0])
```

```
[19, 7, 15, 10, 2, 20, 4, 22, 5, 2, 21, 7, 21, 2, 12, 7, 10, 23, 3, 10, 2, 5, 23, 3, 20, 3, 10, 3, 13, 2, 13, 7, 9, 4, 11, 13,
2, 6, 10, 15, 16, 17]
```

```
x = np.array(x)
y = np.array(y)
# pad sequences at the beginning of each sequence with 0's
# for example if SEQUENCE_LENGTH=4:
# [[5, 3, 2], [5, 1, 2, 3], [3, 4]]
# will be transformed to:
# [[0, 5, 3, 2], [5, 1, 2, 3], [0, 0, 3, 4]]
x = pad_sequences(x, maxlen=SEQUENCE_LENGTH)
```

```
#y = [label in[label] for label in y ]
#y = to_categorical(y)
y = [ label2int[label] for label in y ]
y = to_categorical(y)
```

```
print(y[0])
```

```
[0. 1.]
```

```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=1/3, random_state=7)
```

```
def get_embedding_vectors(tokenizer, dim=100):
    embedding_index = {}
    with open(f"glove.6B.{dim}d.txt", encoding='utf8') as f:
        for line in tqdm.tqdm(f, "Reading GloVe"):
            values = line.split()
            word = values[0]
            vectors = np.asarray(values[1:], dtype='float32')
            embedding_index[word] = vectors

    word_index = tokenizer.word_index
    embedding_matrix = np.zeros((len(word_index)+1, dim))
    for word, i in word_index.items():
        embedding_vector = embedding_index.get(word)
        if embedding_vector is not None:
            # words not found will be 0s
            embedding_matrix[i] = embedding_vector

    return embedding_matrix
```

```
import tensorflow as tf
def get_model(tokenizer, lstm_units):
    """
    Constructs the model,
    Embedding vectors => LSTM => 2 output Fully-Connected neurons with softmax activation
    """
    # get the GloVe embedding vectors
    embedding_matrix = get_embedding_vectors(tokenizer)
    model = Sequential()
    model.add(Embedding(len(tokenizer.word_index)+1,
              EMBEDDING_SIZE,
              weights=[embedding_matrix],
              trainable=False,
              input_length=SEQUENCE_LENGTH))

    model.add(LSTM(lstm_units, recurrent_dropout=0.2))
    model.add(Dropout(0.3))
    model.add(Dense(2, activation="softmax"))
    # compile as rmsprop optimizer
    # aswell as with recall metric
    model.compile(optimizer="rmsprop", loss="binary_crossentropy",
                  metrics=["accuracy", tf.keras.metrics.Precision(), tf.keras.metrics.Recall()])
    model.summary()
    return model
```

```
model = get_model(tokenizer=tokenizer, lstm_units=128)
```

```
Reading GloVe: 400000it [00:16, 24850.71it/s]
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 100, 100)          7600
_________________________________________________________________
lstm_1 (LSTM)                (None, 128)               117248
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 258
=================================================================
Total params: 125,106
Trainable params: 117,506
Non-trainable params: 7,600
_________________________________________________________________
```

```
model_checkpoint = ModelCheckpoint("Real_or_Fake_news{val_loss:.2f}", save_best_only=True,verbose=1)
# for better visualization
tensorboard = TensorBoard(f"Real_or_Fake_news{time.time()}")
# print our data shapes
print("X_train.shape:", X_train.shape)
print("X_test.shape:", X_test.shape)
print("y_train.shape:", y_train.shape)
print("y_test.shape:", y_test.shape)
# train the model
model.fit(X_train, y_train, validation_data=(X_test, y_test),batch_size=32, epochs=100,verbose=1)
```

```
Epoch 8/100
2658/2658 [==============================] - 9s 3ms/step - loss: 0.5786 - accuracy: 0.6896 - precision: 0.6080 - recall: 0.
6080 - val_loss: 0.6666 - val_accuracy: 0.6060 - val_precision: 0.6112 - val_recall: 0.6112
Epoch 9/100
2658/2658 [==============================] - 9s 3ms/step - loss: 0.5667 - accuracy: 0.6904 - precision: 0.6144 - recall: 0.
6144 - val_loss: 0.5965 - val_accuracy: 0.6602 - val_precision: 0.6180 - val_recall: 0.6180
Epoch 10/100
2658/2658 [==============================] - 9s 3ms/step - loss: 0.5496 - accuracy: 0.6949 - precision: 0.6217 - recall: 0.
6217 - val_loss: 0.7184 - val_accuracy: 0.5962 - val_precision: 0.6236 - val_recall: 0.6236
Epoch 11/100
2658/2658 [==============================] - 9s 3ms/step - loss: 0.5440 - accuracy: 0.7020 - precision: 0.6258 - recall: 0.
6258 - val_loss: 0.6043 - val_accuracy: 0.6632 - val_precision: 0.6285 - val_recall: 0.6285y: 0.6992
Epoch 12/100
2658/2658 [==============================] - 9s 3ms/step - loss: 0.5194 - accuracy: 0.7295 - precision: 0.6325 - recall: 0.
6325 - val_loss: 0.7331 - val_accuracy: 0.5789 - val_precision: 0.6340 - val_recall: 0.6340
Epoch 13/100
2658/2658 [==============================] - 9s 3ms/step - loss: 0.5019 - accuracy: 0.7419 - precision: 0.6365 - recall: 0.
6365 - val_loss: 0.5945 - val_accuracy: 0.6759 - val_precision: 0.6394 - val_recall: 0.6394
Epoch 14/100
```

```
# get the loss and metrics
result = model.evaluate(X_test, y_test)
# extract those
loss = result[0]
accuracy = result[1]
precision = result[2]
recall = result[3]
```

```
1330/1330 [==============================] - 1s 1ms/step
```

Figure 12: Checking Epoch

```
def get_predictions(text):
    sequence = tokenizer.texts_to_sequences([text])
    # pad the sequence
    sequence = pad_sequences(sequence, maxlen=SEQUENCE_LENGTH)
    # get the prediction
    prediction = model.predict(sequence)[0]
    # one-hot encoded vector, revert using np.argmax
    return int2label[np.argmax(prediction)]
```

```
text=str(input("enter the statement: "))
```

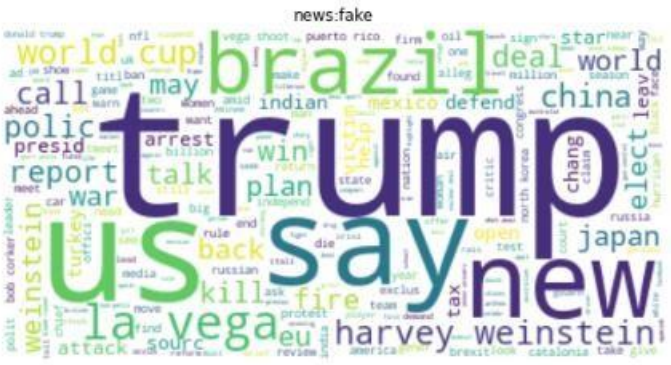
```
enter the statement: Linklater's war veteran comedy speaks to modern America, says star
```

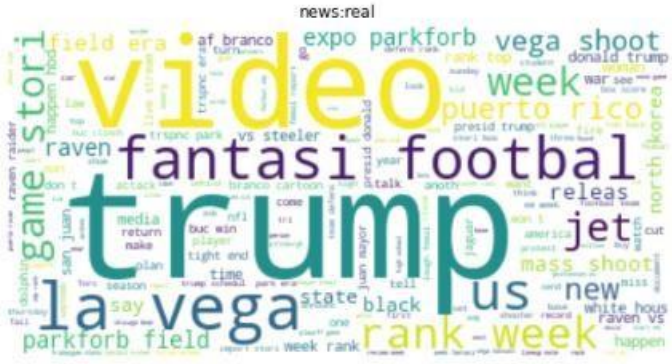
```
print(get_predictions(text))
```

```
Real
```

Figure 13: Prediction of RNN

4 Evaluation Metrics






```
Accuracy result ofDecision Tree Classifier  is: 99.49874686716792


Classification report of Decision Tree Classifier : Results:

              precision    recall  f1-score   support

           0       0.99      1.00      1.00       636
           1       1.00      0.99      0.99       561

    accuracy                           0.99      1197
   macro avg       1.00      0.99      0.99      1197
weighted avg       0.99      0.99      0.99      1197

Confusion Matrix result of Decision Tree Classifier : is:
 [[634   2]
 [  4 557]]

Sensitivity :  0.9968553459119497

Specificity :  0.9928698752228164
```

Figure 14: Evaluation metrics for Decision Tree Classifier

```
 Accuracy result of Logistic Regression is: 81.203007518797


 Classification report of Logistic Regression : Results:

              precision    recall  f1-score   support

           0       0.81      0.84      0.83       636
           1       0.81      0.78      0.79       561

    accuracy                           0.81      1197
   macro avg       0.81      0.81      0.81      1197
weighted avg       0.81      0.81      0.81      1197

 Confusion Matrix result of Logistic Regression : is:
  [[537  99]
  [126 435]]

 Sensitivity :  0.8443396226415094

 Specificity :  0.7754010695187166
```

Figure 15: Evaluation metrics for Logistics Regression

```
Train Score: 0.08 MSE (0.28 RMSE)

Test Score: 0.08 MSE (0.29 RMSE)
```

Figure 15: validation score for Logistics Regression

5 Conclusion`

```
[+] Accuracy: 82.33%
[+] Precision:   86.06%
[+] Recall:   86.06%
```

Figure 16: Conclusion

Consequently, following the same methods as described in the study yields similar results, and it works flawlessly. As a result, the research was a success, and all of the objectives that were set were met.

References