

Fake-Real-News-Detective-Model

October 10, 2021

```
[1]: from bs4 import BeautifulSoup
import re,string,unicodedata

[3]: class clean:
    def __init__(self,text):
        self.text = text

    def strip_html(self,text):
        soup = BeautifulSoup(self.text, "html.parser")
        return soup.get_text()

    # Removing the square brackets
    def remove_betweenn_square_brackets(self, text):
        return re.sub('\[[^\]]*\]', '', self.text)

    # Removing URL's
    def remove_between_square_brackets(self, text):
        return re.sub(r'http\S+', '', self.text)

    # Removing the stopwords from text
    def remove_stopwords(self, text):
        final_text = []
        text = self.text
        for i in text.split():
            if i.strip().lower() not in stop:
                final_text.append(i.strip())
        return " ".join(final_text)

    # Removing the noisy text
    def denoise_text(self, text):
        #text = self.text
        text = self.strip_html(self.text)
        text = self.remove_between_square_brackets(text)
        text = self.remove_stopwords(text)
        return text
```

```
[1]: import nltk
import clean
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import re,string,unicodedata
from collections import Counter
from wordcloud import WordCloud,STOPWORDS
from sklearn.feature_extraction.text import CountVectorizer
```

```
[2]: def get_corpus(text):
    words = []
    for i in text:
        for j in i.split():
            words.append(j.strip())
    return words

def get_top_text_ngrams(corpus, n, g):
    vec = CountVectorizer(ngram_range=(g, g)).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]
```

```
[4]: if __name__ == "__main__":
    # load the real news data and preview
    True_news = pd.read_csv('/Users/yuechenjiang/Desktop/project660/True.csv')
    print('===== Real Data Preview =====')
    print(True_news.head())
    print('===== Real Data Describe =====')
    True_news.describe()
    print('Real Dataset Shape:', '\n', True_news.shape)
    print('Real Columns name', '\n', True_news.columns)
    print('Real Subject count', '\n', True_news['subject'].value_counts())

    # load the fake news data and preview
    Fake_news = pd.read_csv('/Users/yuechenjiang/Desktop/project660/Fake.csv')
    print('===== Fake Data Preview =====')
    print(Fake_news.head())
    print('===== Fake Data Describe =====')
    Fake_news.describe()
    print('Fake Dataset Shape:', '\n', Fake_news.shape)
    print('Fake Columns name', '\n', Fake_news.columns)
    print('Fake Subject count', '\n', Fake_news['subject'].value_counts())
```

```

# combine the datasets and data visualization
True_news['category'] = 1
Fake_news['category'] = 0
df = pd.concat([True_news,Fake_news])
print('===== Comparie the Number of Real and Fake News =====')
sns.set_style("darkgrid")
sns.countplot(df.category)
print('Check whether the data set has null values','\n',df.isna().sum())
print('Check news subjects','\n',df.subject.value_counts())

plt.figure(figsize = (12,8))
plt.title('Real and Fake News subjects')
sns.set(style = "whitegrid",font_scale = 1.2)
chart = sns.countplot(x = "subject", hue = "category" , data = df)
chart.set_xticklabels(chart.get_xticklabels(),rotation=90)

df['text'] = df['text'] + " " + df['title']
del df['title']
del df['subject']
del df['date']

print('===== Real News World Cloud Before Cleaning =====')
plt.figure(figsize = (20,20)) # Text that is not Fake
wc = WordCloud(max_words = 2000 , width = 1600 , height = 800 , stopwords = ↵
↳STOPWORDS).generate(" ".join(df[df.category == 1].text))
plt.imshow(wc , interpolation = 'bilinear')
print('===== Fake News World Cloud Before Cleaning =====')
plt.figure(figsize = (20,20)) # Text that is Fake
wc = WordCloud(max_words = 2000 , width = 1600 , height = 800 , stopwords = ↵
↳STOPWORDS).generate(" ".join(df[df.category == 0].text))
plt.imshow(wc , interpolation = 'bilinear')

clean_text = []
for i in df['text']:
    data_cleaning = clean.clean(text = i)
    clean_text.append(data_cleaning.denoise_text(i))
del df['text']
df = pd.DataFrame({'text':clean_text,'category':df['category']})
# df.head()
# data_cleaning = clean()
# df['text'] = df['text'].apply(data_cleaning.denoise_text)
# WHEN I USE CLASS I'M UNABLE TO USE 'apply' FUNCTION, STILL NEED TO FIND ↵
↳OUT WHY, USING LOOPS ARE SLOW
print('===== Real News World Cloud After Cleaning =====')
plt.figure(figsize = (20,20)) # Text that is not Fake

```

```

wc = WordCloud(max_words = 2000 , width = 1600 , height = 800 , stopwords =
↳STOPWORDS).generate(" ".join(df[df.category == 1].text))
plt.imshow(wc , interpolation = 'bilinear')
print('===== Fake News World Cloud After Cleaning =====')
plt.figure(figsize = (20,20)) # Text that is Fake
wc = WordCloud(max_words = 2000 , width = 1600 , height = 800 , stopwords =
↳STOPWORDS).generate(" ".join(df[df.category == 0].text))
plt.imshow(wc , interpolation = 'bilinear')
# Number of characters in texts
fig,(ax1,ax2)=plt.subplots(1,2,figsize=(12,8))
text_len=df[df['category']==1]['text'].str.len()
ax1.hist(text_len,color='red')
ax1.set_title('Original text')
text_len=df[df['category']==0]['text'].str.len()
ax2.hist(text_len,color='green')
ax2.set_title('Fake text')
fig.suptitle('Characters in texts')
plt.show()
print('The distribution of both seems to be a bit different.','\n',
      '2500 characters in text is the most common in original text_
↳category', '\n',
      'while around 5000 characters in text are most common in fake text_
↳category.')

# Number of words in each text
fig,(ax1,ax2)=plt.subplots(1,2,figsize=(12,8))
text_len=df[df['category']==1]['text'].str.split().map(lambda x: len(x))
ax1.hist(text_len,color='red')
ax1.set_title('Original text')
text_len=df[df['category']==0]['text'].str.split().map(lambda x: len(x))
ax2.hist(text_len,color='green')
ax2.set_title('Fake text')
fig.suptitle('Words in texts')
plt.show()

# Average word length in a text
fig,(ax1,ax2)=plt.subplots(1,2,figsize=(20,10))
word=df[df['category']==1]['text'].str.split().apply(lambda x : [len(i) for
↳i in x])
sns.distplot(word.map(lambda x: np.mean(x)),ax=ax1,color='red')
ax1.set_title('Original text')
word=df[df['category']==0]['text'].str.split().apply(lambda x : [len(i) for
↳i in x])
sns.distplot(word.map(lambda x: np.mean(x)),ax=ax2,color='green')
ax2.set_title('Fake text')
fig.suptitle('Average word length in each text')

```

```

corpus = get_corpus(df.text)
print('Top 5 Words', '\n', corpus[:5])

counter = Counter(corpus)
most_common = counter.most_common(10)
most_common = dict(most_common)
print('Numbers of most common words', '\n', most_common)

# Unigram Analysis
plt.figure(figsize = (16,9))
most_common_uni = get_top_text_ngrams(df.text,10,1)
most_common_uni = dict(most_common_uni)
sns.barplot(x=list(most_common_uni.values()),y=list(most_common_uni.keys()))

# Bigram Analysis
plt.figure(figsize = (16,9))
most_common_bi = get_top_text_ngrams(df.text,10,2)
most_common_bi = dict(most_common_bi)
sns.barplot(x=list(most_common_bi.values()),y=list(most_common_bi.keys()))

# Trigram Analysis
plt.figure(figsize = (16,9))
most_common_tri = get_top_text_ngrams(df.text,10,3)
most_common_tri = dict(most_common_tri)
sns.barplot(x=list(most_common_tri.values()),y=list(most_common_tri.keys()))

```

===== Real Data Preview =====

| | title \ | text | subject \ | date |
|---|---|------|--------------|-------------------|
| 0 | As U.S. budget fight looms, Republicans flip t... | | politicsNews | December 31, 2017 |
| 1 | U.S. military to accept transgender recruits o... | | politicsNews | December 29, 2017 |
| 2 | Senior U.S. Republican senator: 'Let Mr. Muell... | | politicsNews | December 31, 2017 |
| 3 | FBI Russia probe helped by Australian diplomat... | | politicsNews | December 30, 2017 |
| 4 | Trump wants Postal Service to charge 'much mor... | | politicsNews | |

```

4 December 29, 2017
===== Real Data Describe =====
Real Dataset Shape:
(21417, 4)
Real Columns name
Index(['title', 'text', 'subject', 'date'], dtype='object')
Real Subject count
politicsNews      11272
worldnews         10145
Name: subject, dtype: int64
===== Fake Data Preview =====

                                title \
0 Donald Trump Sends Out Embarrassing New Year'...
1 Drunk Bragging Trump Staffer Started Russian ...
2 Sheriff David Clarke Becomes An Internet Joke...
3 Trump Is So Obsessed He Even Has Obama's Name...
4 Pope Francis Just Called Out Donald Trump Dur...

                                text subject \
0 Donald Trump just couldn t wish all Americans ... News
1 House Intelligence Committee Chairman Devin Nu... News
2 On Friday, it was revealed that former Milwauk... News
3 On Christmas day, Donald Trump announced that ... News
4 Pope Francis used his annual Christmas Day mes... News

                                date
0 December 31, 2017
1 December 31, 2017
2 December 30, 2017
3 December 29, 2017
4 December 25, 2017
===== Fake Data Describe =====
Fake Dataset Shape:
(23481, 4)
Fake Columns name
Index(['title', 'text', 'subject', 'date'], dtype='object')
Fake Subject count
News      9050
politics  6841
left-news 4459
Government News 1570
US_News   783
Middle-east 778
Name: subject, dtype: int64
===== Comparie the Number of Real and Fake News =====
Check whether the data set has null values
title      0
text       0

```

```

subject      0
date         0
category     0
dtype: int64
Check news subjects
  politicsNews      11272
worldnews          10145
News               9050
politics           6841
left-news          4459
Government News    1570
US_News            783
Middle-east        778

```

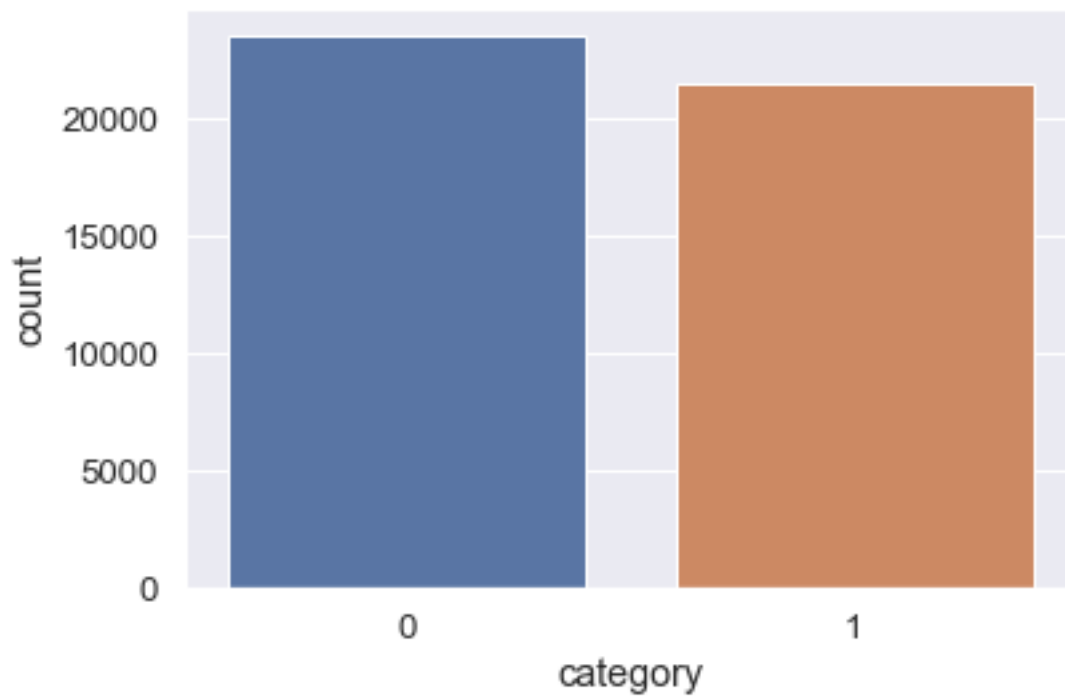
```
Name: subject, dtype: int64
```

```
===== Real News World Cloud Before Cleaning =====
```

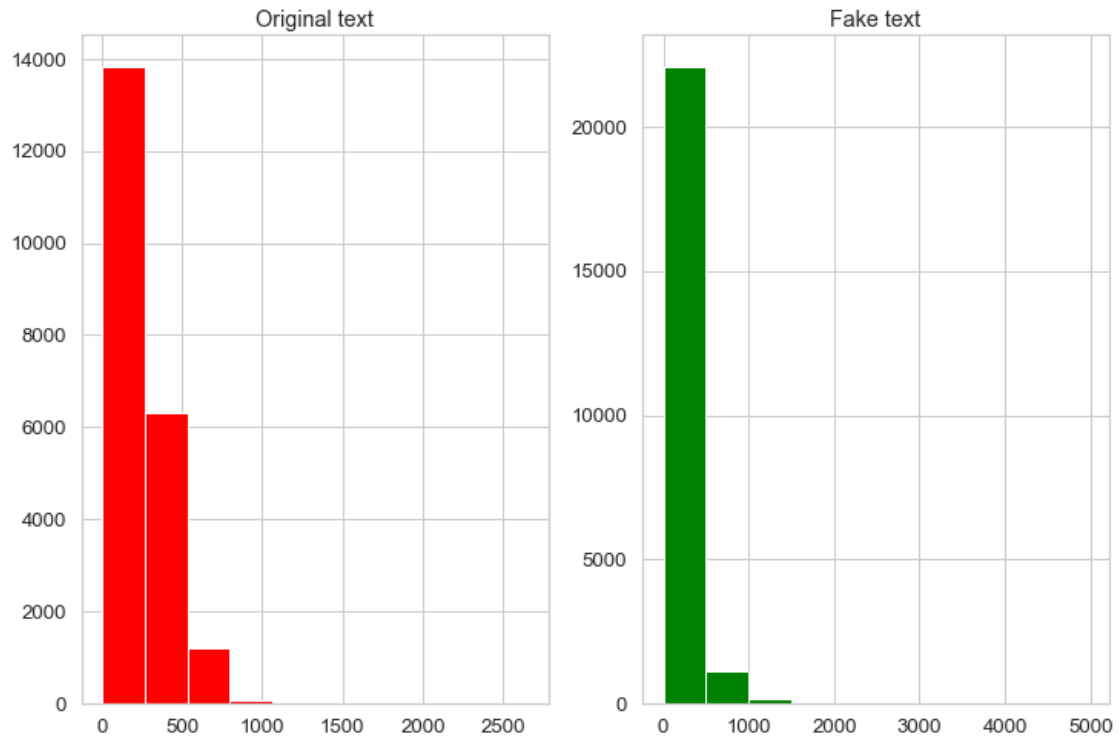
```
===== Fake News World Cloud Before Cleaning =====
```

```
===== Real News World Cloud After Cleaning =====
```

```
===== Fake News World Cloud After Cleaning =====
```



Words in texts

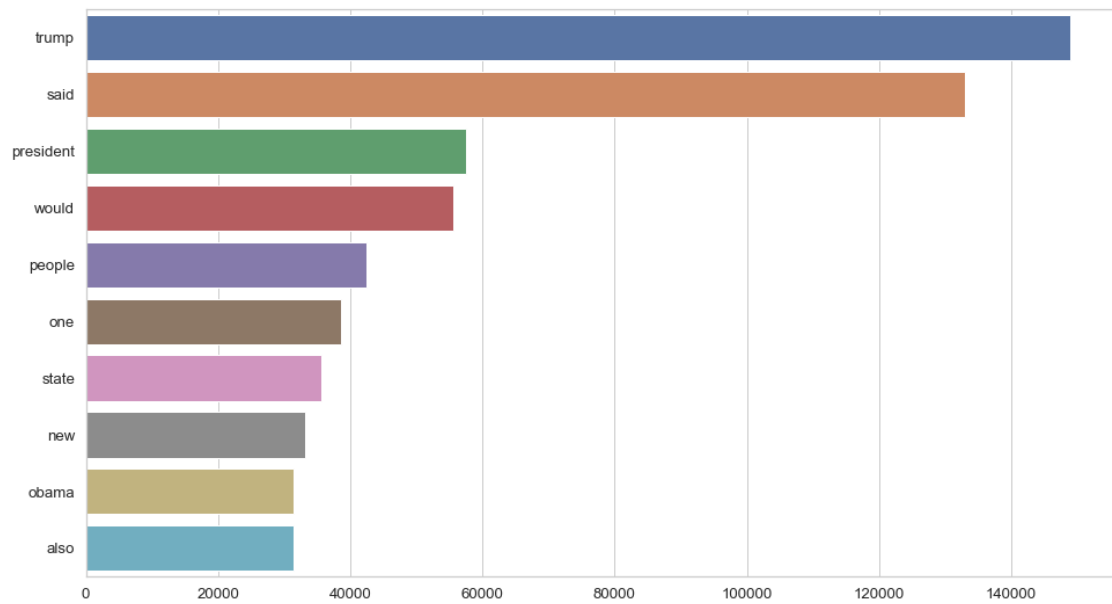
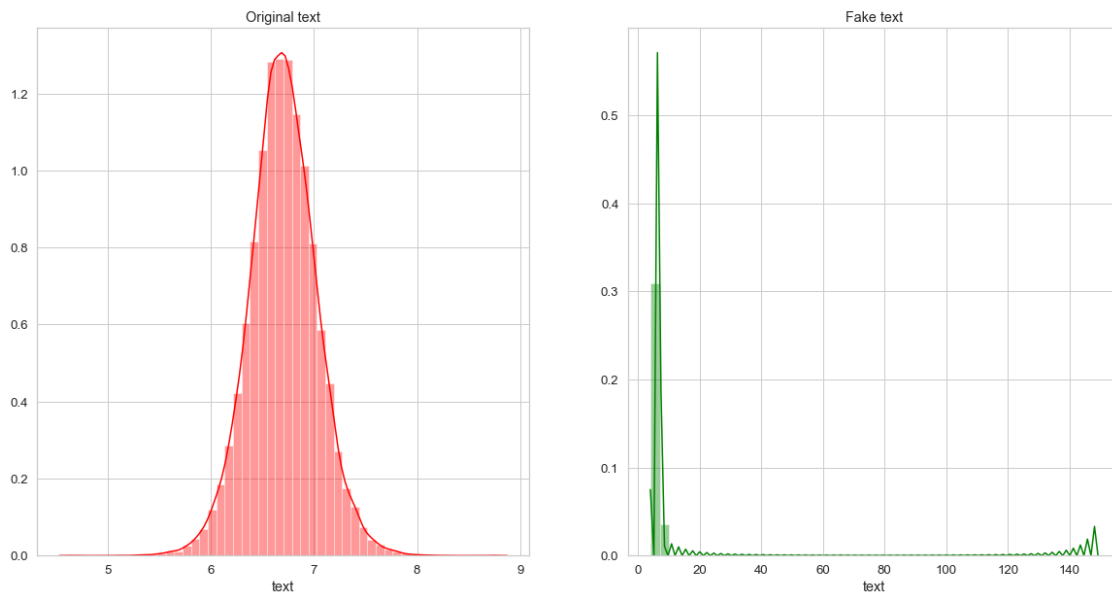


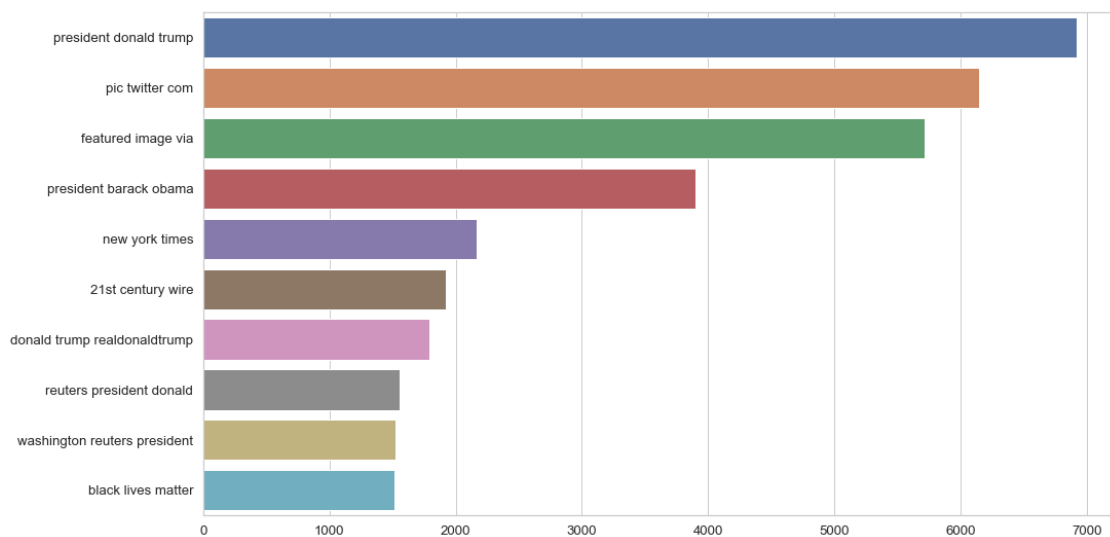
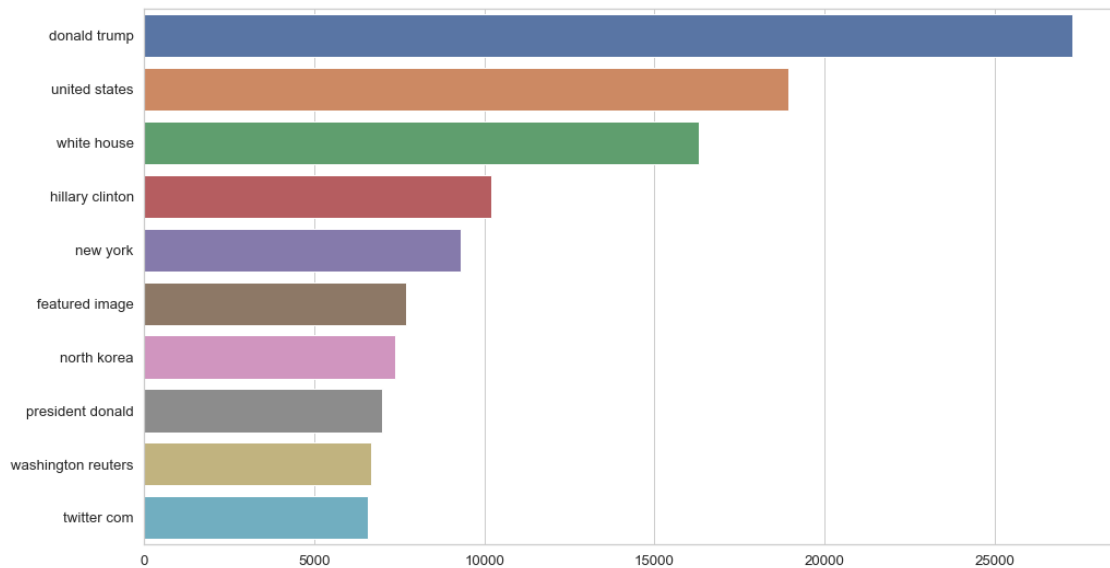
Top 5 Words

['WASHINGTON', '(Reuters)', 'head', 'conservative', 'Republican']

Numbers of most common words {'Trump': 111503, 'said': 93162, 'would': 54613, 'U.S.': 50441, 'President': 33180, 'people': 33115, 'also': 30325, 'one': 29370, 'Donald': 27795, 'said.': 26190}

Average word length in each text





```
[7]: stop = set(stopwords.words('english'))
punctuation = list(string.punctuation)
stop.update(punctuation)
```

```
[38]: def strip_html(text):
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

    #Removing the square brackets
    def remove_between_square_brackets(text):
```

```

        return re.sub('\[[^\]]*\]', '', text)
# Removing URL's
def remove_between_square_brackets(text):
    return re.sub(r'http\S+', '', text)
#Removing the stopwords from text
def remove_stopwords(text):
    final_text = []
    for i in text.split():
        if i.strip().lower() not in stop:
            final_text.append(i.strip())
    return " ".join(final_text)
#Removing the noisy text
def denoise_text(text):
    text = strip_html(text)
    text = remove_between_square_brackets(text)
    text = remove_stopwords(text)
    return text

```

```

[2]: True_news = pd.read_csv('/Users/yuechenjiang/Desktop/project660/True.csv')
Fake_news = pd.read_csv('/Users/yuechenjiang/Desktop/project660/Fake.csv')
True_news['category'] = 1
Fake_news['category'] = 0
df = pd.concat([True_news,Fake_news])
df['text']=df['text'].apply(denoise_text)

```

```

[5]: import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn import model_selection
from sklearn import preprocessing
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics

```

```
[6]: True_news = pd.read_csv('/Users/yuechenjiang/Desktop/project660/True.csv')
Fake_news = pd.read_csv('/Users/yuechenjiang/Desktop/project660/Fake.csv')
True_news['category'] = 1
Fake_news['category'] = 0
df = pd.concat([True_news,Fake_news])
df['text'] = df['text'] + " " + df['title']
del df['title']
del df['subject']
del df['date']
clean_text = []
for i in df['text']:
    data_cleaning = clean.clean(text = i)
    clean_text.append(data_cleaning.denoise_text(i))
del df['text']
df = pd.DataFrame({'text':clean_text,'category':df['category']})
```

```
[7]: models = [LogisticRegression(solver='lbfgs'),           # Logistic regression
               RandomForestClassifier(n_estimators=100),      # Random forest
               DecisionTreeClassifier(),                      # Decision tree
               MLPClassifier(max_iter=100),                  # Multilayer perceptron
               AdaBoostClassifier(),                        # Adaptive gradient boost
               BaggingClassifier(),                         # Bagging algorithm
               GradientBoostingClassifier(),                # Gradient Boosting
               ↪Algorithm
               SVC(kernel = 'linear')]
               #GaussianNB()]

model_name = ['LogisticRegression',
              'RandomForestClassifier',
              'DecisionTreeClassifier',
              'MLPClassifier',
              'AdaBoostClassifier',
              'BaggingClassifier',
              'GradientBoostingClassifier',
              'SVMClassifier']
```

```
[8]: tfidf_vectorizer = TfidfVectorizer(use_idf=True, stop_words='english')
X = tfidf_vectorizer.fit_transform(df['text'])
Y = df['category']
x_train, x_test, y_train, y_test = model_selection.train_test_split(X, Y ,
    ↪train_size=0.8,test_size=0.2, random_state=1)
```

```
[18]: #fig, (ax1,ax2)=plt.subplots(figsize=(20,8))
acc = []
cms = []
for model in models:
    model.fit(x_train,y_train)
```



```

# model_acc = model.score(x_test, y_test)*100
acc.append(model.score(x_test, y_test))
y_pred = model.predict(x_test)
cm = confusion_matrix(y_test,y_pred)
cms.append(cm)
print(model, '\n', cm)
#sns.heatmap(cm,cmap= "Blues", linecolor = 'black' , linewidth = 1 , annot_
→= True, fmt=' ' , xticklabels = ['Fake','Original'] , yticklabels =
→['Fake','Original'])
#plt.xlabel("Predicted")
#plt.ylabel("Actual")

```

```

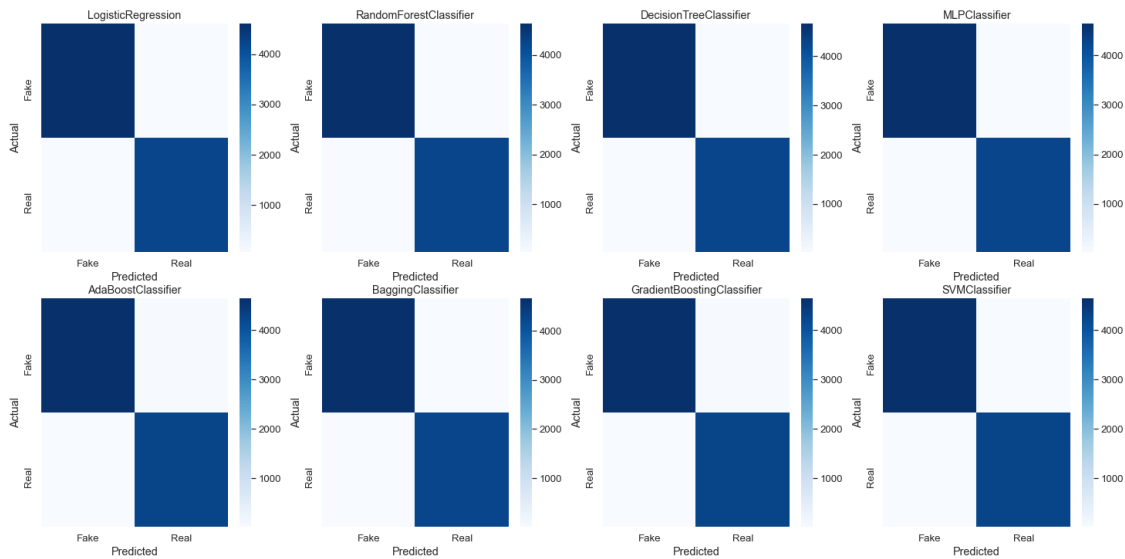
LogisticRegression()
[[4607  71]
 [ 56 4246]]
RandomForestClassifier()
[[4629  49]
 [ 37 4265]]
DecisionTreeClassifier()
[[4660  18]
 [ 18 4284]]
MLPClassifier(max_iter=100)
[[4642  36]
 [ 23 4279]]
AdaBoostClassifier()
[[4648  30]
 [  8 4294]]
BaggingClassifier()
[[4661  17]
 [  9 4293]]
GradientBoostingClassifier()
[[4644  34]
 [ 10 4292]]
SVC(kernel='linear')
[[4646  32]
 [ 15 4287]]

```

```

[28]: fig,ax=plt.subplots(2,4,figsize=(25,12))
for i in range(len(cms)):
    plt.subplot(2, 4, i+1)
    sns.heatmap(np.array(cms[i]),cmap= "Blues", linecolor = 'black' ,
→xticklabels = ['Fake','Real'] , yticklabels = ['Fake','Real'])
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title(model_name[i])

```

```
[20]: a = pd.DataFrame({"name": model_name, "acc": acc})
a
```

```
[20]:
```

| | name | acc |
|---|----------------------------|----------|
| 0 | LogisticRegression | 0.985857 |
| 1 | RandomForestClassifier | 0.990423 |
| 2 | DecisionTreeClassifier | 0.995991 |
| 3 | MLPClassifier | 0.993430 |
| 4 | AdaBoostClassifier | 0.995768 |
| 5 | BaggingClassifier | 0.997105 |
| 6 | GradientBoostingClassifier | 0.995100 |
| 7 | SVMClassifier | 0.994766 |

```
[1]: import os
import clean
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import nltk
from sklearn.preprocessing import LabelBinarizer
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from wordcloud import WordCloud, STOPWORDS
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize, sent_tokenize
from bs4 import BeautifulSoup
import re, string, unicodedata
```

```

from keras.preprocessing import text, sequence
from sklearn.metrics import
    ↳classification_report, confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split
from string import punctuation
from nltk import pos_tag
from nltk.corpus import wordnet
from tensorflow import keras
from tensorflow.keras import layers
import keras
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, Dropout
from keras.callbacks import ReduceLROnPlateau
import tensorflow as tf
from collections import Counter
from sklearn.feature_extraction.text import CountVectorizer

```

```

[3]: True_news = pd.read_csv('/Users/yuechenjiang/Desktop/project660/code&data/True.
    ↳csv')
Fake_news = pd.read_csv('/Users/yuechenjiang/Desktop/project660/code&data/Fake.
    ↳csv')
True_news['category'] = 1
Fake_news['category'] = 0
df = pd.concat([True_news, Fake_news])
df['text'] = df['text'] + " " + df['title']
del df['title']
del df['subject']
del df['date']
clean_text = []
for i in df['text']:
    data_cleaning = clean.clean(text = i)
    clean_text.append(data_cleaning.denoise_text(i))
del df['text']
df = pd.DataFrame({'text':clean_text, 'category':df['category']})

```

```

[4]: df.to_csv("combined.csv", index=False)

```

```

[2]: df = pd.read_csv('/Users/yuechenjiang/Desktop/project660/code&data/combined.
    ↳csv')

```

```

[17]: x_train, x_test, y_train, y_test = train_test_split(df.text, df.
    ↳category, random_state = 0)

max_features = 10000
maxlen = 101

# Tokenizing Text -> Representing each word by a number

```

```

# Mapping of original word to number is preserved in word_index property of
↳ tokenizer
# Tokenizer applies basic processing like changing it to lower case,
↳ explicitly setting that as False
# Lets keep all news to 300, add padding to news with less than 300 words and
↳ truncating long ones
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)

tokenized_test = tokenizer.texts_to_sequences(x_test)
X_test = sequence.pad_sequences(tokenized_test, maxlen=maxlen)

```

```
[18]: np.savetxt('train.txt', x_train)
```

```
[19]: EMBEDDING_FILE = '/Users/yuechenjiang/Desktop/project660/code&data/train.txt'
```

```

[20]: def get_coefs(word, *arr):
        return word, np.asarray(arr, dtype='float32')
embeddings_index = dict(get_coefs(*o.rstrip().rsplit(' ')) for o in
↳ open(EMBEDDING_FILE))

```

```

[21]: all_embs = np.stack(embeddings_index.values())
emb_mean, emb_std = all_embs.mean(), all_embs.std()
embed_size = all_embs.shape[1]

```

```

[22]: word_index = tokenizer.word_index
nb_words = min(max_features, len(word_index))
# change below line if computing normal stats is too slow
embedding_matrix = np.random.normal(emb_mean, emb_std, (nb_words, embed_size))
for word, i in word_index.items():
    if i >= max_features: continue
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None: embedding_matrix[i] = embedding_vector

```

```
[23]: embed_size
```

```
[23]: 100
```

```
[24]: embedding_matrix.shape
```

```
[24]: (10000, 100)
```

```
[25]: emb_mean
```

```
[25]: 1615.2051
```

```
[30]: # Model Parameters
batch_size = 256
epochs = 12
embed_size = 100
```

```
[27]: learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy', patience = 2,
↳ verbose=1, factor=0.5, min_lr=0.00001)
```

```
[28]: from tensorflow import keras
from tensorflow.keras import layers
#Defining Neural Network
model = Sequential()
#Non-trainable embedding layer
model.add(Embedding(max_features, output_dim=embed_size,
↳ weights=[embedding_matrix], input_length=300, 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(learning_rate = 0.01),
↳ loss='binary_crossentropy', metrics=['accuracy'])
```

```
[29]: model.summary()
```

Model: "sequential_3"

| Layer (type) | Output Shape | Param # |
|-------------------------|------------------|---------|
| embedding_1 (Embedding) | (None, 300, 100) | 1000000 |
| lstm (LSTM) | (None, 300, 128) | 117248 |
| lstm_1 (LSTM) | (None, 64) | 49408 |
| dense (Dense) | (None, 32) | 2080 |
| dense_1 (Dense) | (None, 1) | 33 |

Total params: 1,168,769
Trainable params: 168,769
Non-trainable params: 1,000,000

```
[31]: history = model.fit(x_train, y_train, batch_size = batch_size , validation_data=
↳ (X_test,y_test) , epochs = epochs , callbacks = [learning_rate_reduction])
```

Epoch 1/12
WARNING:tensorflow:Model was constructed with shape (None, 300) for input KerasTensor(type_spec=TensorSpec(shape=(None, 300), dtype=tf.float32, name='embedding_1_input'), name='embedding_1_input', description="created by layer 'embedding_1_input'"), but it was called on an input with incompatible shape (None, 101).
WARNING:tensorflow:Model was constructed with shape (None, 300) for input KerasTensor(type_spec=TensorSpec(shape=(None, 300), dtype=tf.float32, name='embedding_1_input'), name='embedding_1_input', description="created by layer 'embedding_1_input'"), but it was called on an input with incompatible shape (None, 101).
132/132 [=====] - ETA: 0s - loss: 0.5488 - accuracy: 0.6789
WARNING:tensorflow:Model was constructed with shape (None, 300) for input KerasTensor(type_spec=TensorSpec(shape=(None, 300), dtype=tf.float32, name='embedding_1_input'), name='embedding_1_input', description="created by layer 'embedding_1_input'"), but it was called on an input with incompatible shape (None, 101).
132/132 [=====] - 101s 740ms/step - loss: 0.5488 - accuracy: 0.6789 - val_loss: 0.2801 - val_accuracy: 0.8861
Epoch 2/12
132/132 [=====] - 118s 894ms/step - loss: 0.4190 - accuracy: 0.7988 - val_loss: 0.3402 - val_accuracy: 0.8632
Epoch 3/12
132/132 [=====] - 113s 855ms/step - loss: 0.3375 - accuracy: 0.8459 - val_loss: 0.2524 - val_accuracy: 0.8885
Epoch 4/12
132/132 [=====] - 137s 1s/step - loss: 0.2912 - accuracy: 0.8714 - val_loss: 0.2506 - val_accuracy: 0.8927
Epoch 5/12
132/132 [=====] - 147s 1s/step - loss: 0.2707 - accuracy: 0.8821 - val_loss: 0.2542 - val_accuracy: 0.8900
Epoch 6/12
132/132 [=====] - 176s 1s/step - loss: 0.2739 - accuracy: 0.8819 - val_loss: 0.2597 - val_accuracy: 0.8893

Epoch 00006: ReduceLROnPlateau reducing learning rate to 0.004999999888241291.
Epoch 7/12
132/132 [=====] - 160s 1s/step - loss: 0.2613 - accuracy: 0.8854 - val_loss: 0.2323 - val_accuracy: 0.9040
Epoch 8/12
132/132 [=====] - 142s 1s/step - loss: 0.2442 - accuracy: 0.8948 - val_loss: 0.2034 - val_accuracy: 0.9146
Epoch 9/12
132/132 [=====] - 138s 1s/step - loss: 0.2432 - accuracy: 0.8959 - val_loss: 0.2294 - val_accuracy: 0.9057
Epoch 10/12
132/132 [=====] - 139s 1s/step - loss: 0.2349 - accuracy: 0.8981 - val_loss: 0.2230 - val_accuracy: 0.9067

Epoch 00010: ReduceLROnPlateau reducing learning rate to 0.0024999999441206455.

Epoch 11/12

132/132 [=====] - 132s 999ms/step - loss: 0.2230 - accuracy: 0.9054 - val_loss: 0.1981 - val_accuracy: 0.9132

Epoch 12/12

132/132 [=====] - 152s 1s/step - loss: 0.2147 - accuracy: 0.9096 - val_loss: 0.1822 - val_accuracy: 0.9261

```
[32]: # Model Analysis
print("Accuracy of the model on Training Data is - " , model.
      ↪evaluate(x_train,y_train)[1]*100 , "%")
print("Accuracy of the model on Testing Data is - " , model.
      ↪evaluate(X_test,y_test)[1]*100 , "%")
```

1053/1053 [=====] - 44s 42ms/step - loss: 0.1856 - accuracy: 0.9272

Accuracy of the model on Training Data is - 92.72413849830627 %

351/351 [=====] - 15s 44ms/step - loss: 0.1822 - accuracy: 0.9261

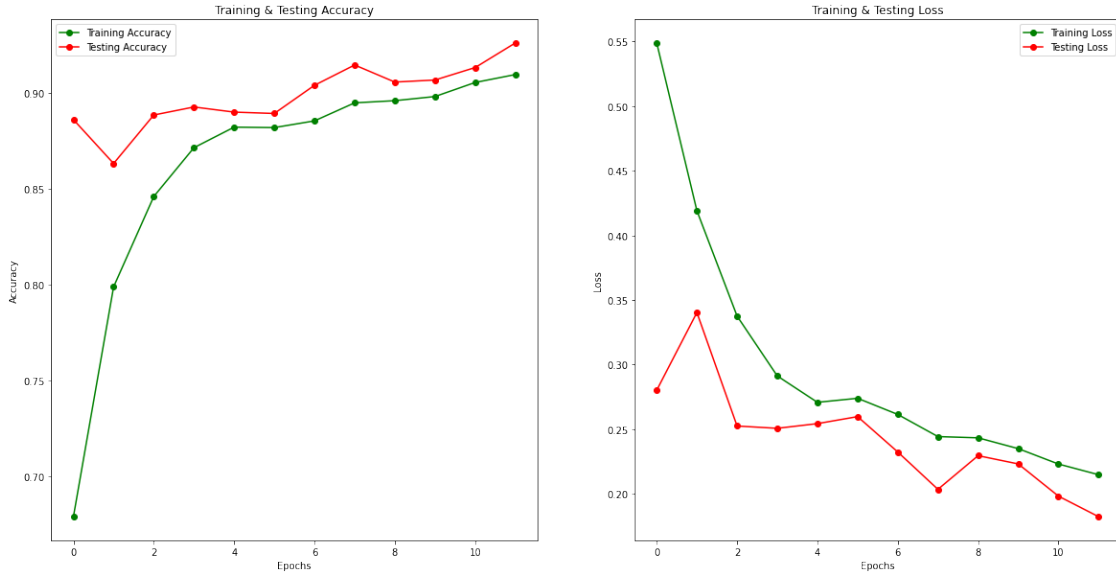
Accuracy of the model on Testing Data is - 92.60579347610474 %

```
[33]: epochs = [i for i in range(12)]
fig , ax = plt.subplots(1,2)
train_acc = history.history['accuracy']
train_loss = history.history['loss']
val_acc = history.history['val_accuracy']
val_loss = history.history['val_loss']

fig.set_size_inches(20,10)

ax[0].plot(epochs , train_acc , 'go-' , label = 'Training Accuracy')
ax[0].plot(epochs , val_acc , 'ro-' , label = 'Testing Accuracy')
ax[0].set_title('Training & Testing Accuracy')
ax[0].legend()
ax[0].set_xlabel("Epochs")
ax[0].set_ylabel("Accuracy")

ax[1].plot(epochs , train_loss , 'go-' , label = 'Training Loss')
ax[1].plot(epochs , val_loss , 'ro-' , label = 'Testing Loss')
ax[1].set_title('Training & Testing Loss')
ax[1].legend()
ax[1].set_xlabel("Epochs")
ax[1].set_ylabel("Loss")
plt.show()
```



```
[34]: pred = model.predict(X_test)
      pred[:5]
```

WARNING:tensorflow:Model was constructed with shape (None, 300) for input KerasTensor(type_spec=TensorSpec(shape=(None, 300), dtype=tf.float32, name='embedding_1_input'), name='embedding_1_input', description="created by layer 'embedding_1_input'"), but it was called on an input with incompatible shape (None, 101).

```
[34]: array([[1.6572118e-02],
             [1.2124286e-05],
             [7.2982348e-06],
             [1.0700109e-04],
             [9.8854536e-01]], dtype=float32)
```

```
[35]: pred = list(pred)
      test_result = []
      for i in range(len(pred)):
          test_result.append(pred[i][0])
```

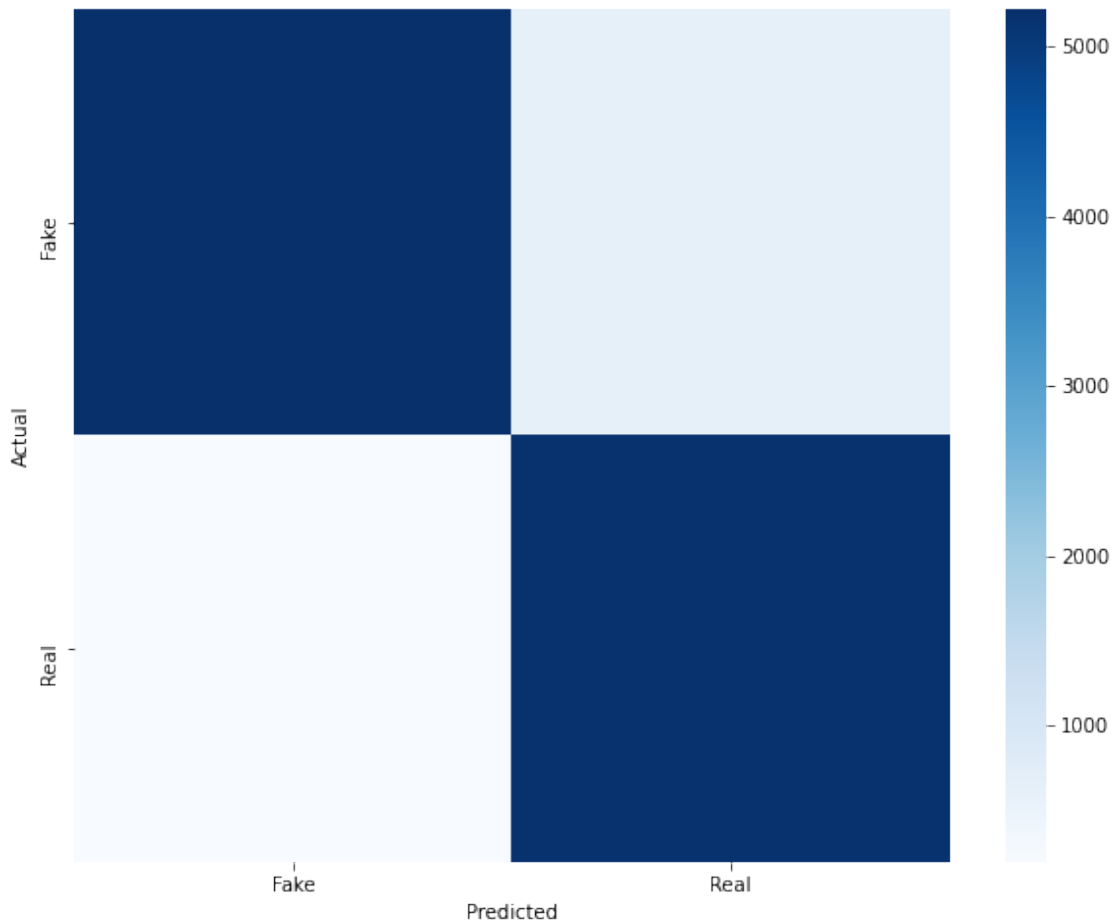
```
[36]: confusion = pd.DataFrame({'Pred':test_result, 'Truth':list(y_test)})
      confusion['binary_pred'] = (confusion['Pred'] > 0.5).astype(int)
```

```
[37]: cm_DL = confusion_matrix(confusion['Truth'],confusion['binary_pred'])
      cm_DL
```

```
[37]: array([[5221,  637],
            [ 193, 5174]])
```

```
[38]: plt.figure(figsize = (10,8))
sns.heatmap(cm_DL,cmap= "Blues", linecolor = 'black' , xticklabels = ['Fake', 'Real'] , yticklabels = ['Fake', 'Real'])
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

```
[38]: Text(69.0, 0.5, 'Actual')
```



```
[ ]: from tensorflow import keras
from tensorflow.keras import layers
#Defining Neural Network
model = Sequential()
#Non-trainable embedding layer
model.add(Embedding(max_features, output_dim=embed_size,
weights=[embedding_matrix], input_length=100, 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(learning_rate = 0.01),
↳loss='binary_crossentropy', metrics=['accuracy'])

```

```

[16]: history = model.fit(x_train, y_train, batch_size = batch_size , validation_data
↳= (X_test,y_test) , epochs = epochs , callbacks = [learning_rate_reduction])

```

```

Epoch 1/15
132/132 [=====] - 158s 1s/step - loss: 0.5791 -
accuracy: 0.6556 - val_loss: 0.5158 - val_accuracy: 0.7183
Epoch 2/15
132/132 [=====] - 122s 924ms/step - loss: 0.4687 -
accuracy: 0.7630 - val_loss: 0.3673 - val_accuracy: 0.8392
Epoch 3/15
132/132 [=====] - 127s 962ms/step - loss: 0.4201 -
accuracy: 0.8005 - val_loss: 0.3459 - val_accuracy: 0.8577
Epoch 4/15
132/132 [=====] - 125s 941ms/step - loss: 0.3738 -
accuracy: 0.8273 - val_loss: 0.3302 - val_accuracy: 0.8477
Epoch 5/15
132/132 [=====] - 125s 947ms/step - loss: 0.3471 -
accuracy: 0.8413 - val_loss: 0.2792 - val_accuracy: 0.8845
Epoch 6/15
132/132 [=====] - 129s 976ms/step - loss: 0.3673 -
accuracy: 0.8322 - val_loss: 0.3263 - val_accuracy: 0.8744
Epoch 7/15
132/132 [=====] - 123s 923ms/step - loss: 0.3494 -
accuracy: 0.8390 - val_loss: 0.2641 - val_accuracy: 0.8924
Epoch 8/15
132/132 [=====] - 157s 1s/step - loss: 0.3034 -
accuracy: 0.8628 - val_loss: 0.2489 - val_accuracy: 0.8930
Epoch 9/15
132/132 [=====] - 164s 1s/step - loss: 0.2839 -
accuracy: 0.8694 - val_loss: 0.2425 - val_accuracy: 0.8892
Epoch 10/15
132/132 [=====] - 162s 1s/step - loss: 0.2828 -
accuracy: 0.8719 - val_loss: 0.2071 - val_accuracy: 0.9070
Epoch 11/15
132/132 [=====] - 230s 2s/step - loss: 0.2639 -
accuracy: 0.8803 - val_loss: 0.1999 - val_accuracy: 0.9118
Epoch 12/15
132/132 [=====] - 139s 1s/step - loss: 0.2688 -
accuracy: 0.8764 - val_loss: 0.2166 - val_accuracy: 0.9114
Epoch 13/15
132/132 [=====] - 158s 1s/step - loss: 0.2638 -

```

accuracy: 0.8769 - val_loss: 0.2275 - val_accuracy: 0.8959

Epoch 00013: ReduceLROnPlateau reducing learning rate to 0.004999999888241291.

Epoch 14/15

132/132 [=====] - 126s 950ms/step - loss: 0.2575 -

accuracy: 0.8818 - val_loss: 0.1948 - val_accuracy: 0.9133

Epoch 15/15

132/132 [=====] - 125s 950ms/step - loss: 0.2390 -

accuracy: 0.8895 - val_loss: 0.2193 - val_accuracy: 0.8951

```
[17]: # Model Analysis
print("Accuracy of the model on Training Data is - " , model.
      ↪evaluate(x_train,y_train)[1]*100 , "%")
print("Accuracy of the model on Testing Data is - " , model.
      ↪evaluate(X_test,y_test)[1]*100 , "%")
```

1053/1053 [=====] - 43s 41ms/step - loss: 0.2183 -
accuracy: 0.8983

Accuracy of the model on Training Data is - 89.82864618301392 %

351/351 [=====] - 14s 40ms/step - loss: 0.2193 -

accuracy: 0.8951

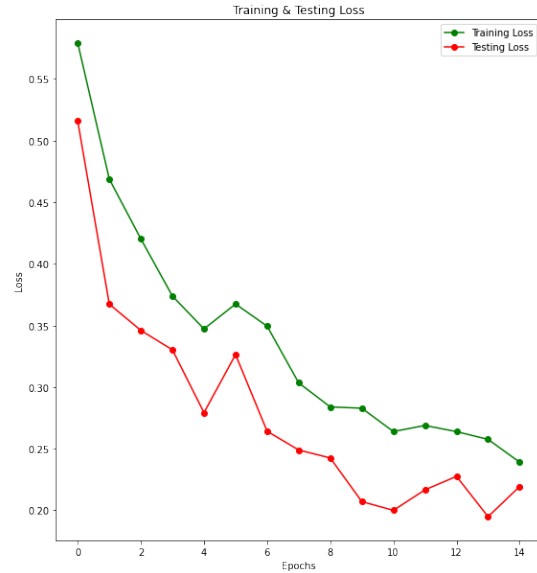
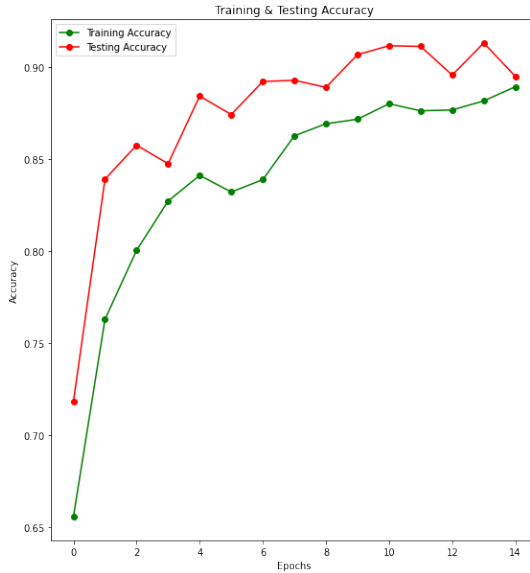
Accuracy of the model on Testing Data is - 89.51447606086731 %

```
[18]: epochs = [i for i in range(15)]
fig , ax = plt.subplots(1,2)
train_acc = history.history['accuracy']
train_loss = history.history['loss']
val_acc = history.history['val_accuracy']
val_loss = history.history['val_loss']

fig.set_size_inches(20,10)

ax[0].plot(epochs , train_acc , 'go-' , label = 'Training Accuracy')
ax[0].plot(epochs , val_acc , 'ro-' , label = 'Testing Accuracy')
ax[0].set_title('Training & Testing Accuracy')
ax[0].legend()
ax[0].set_xlabel("Epochs")
ax[0].set_ylabel("Accuracy")

ax[1].plot(epochs , train_loss , 'go-' , label = 'Training Loss')
ax[1].plot(epochs , val_loss , 'ro-' , label = 'Testing Loss')
ax[1].set_title('Training & Testing Loss')
ax[1].legend()
ax[1].set_xlabel("Epochs")
ax[1].set_ylabel("Loss")
plt.show()
```



```
[19]: pred = model.predict(X_test)
      pred[:5]
```

```
[19]: array([[8.5334843e-01],
             [6.8051787e-07],
             [3.1262412e-06],
             [5.1984191e-04],
             [9.9682760e-01]], dtype=float32)
```

```
[20]: pred = list(pred)
      test_result = []
      for i in range(len(pred)):
          test_result.append(pred[i][0])
```

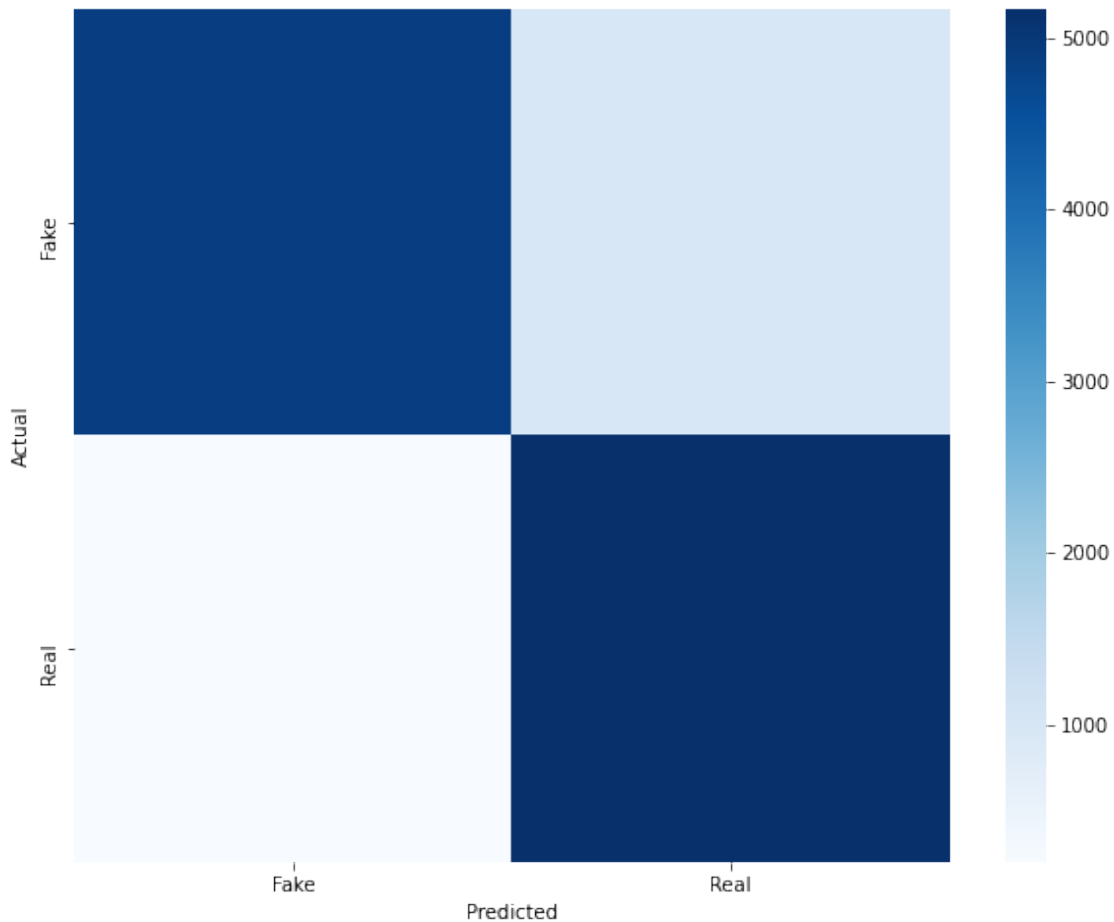
```
[21]: confusion = pd.DataFrame({'Pred':test_result, 'Truth':list(y_test)})
      confusion['binary_pred'] = (confusion['Pred'] > 0.5).astype(int)
```

```
[22]: cm_DL = confusion_matrix(confusion['Truth'],confusion['binary_pred'])
      cm_DL
```

```
[22]: array([[4881,  977],
            [ 200, 5167]])
```

```
[23]: plt.figure(figsize = (10,8))
      sns.heatmap(cm_DL,cmap= "Blues", linecolor = 'black' , xticklabels = _
      ↪ ['Fake','Real'] , yticklabels = ['Fake','Real'])
      plt.xlabel("Predicted")
      plt.ylabel("Actual")
```

[23]: Text(69.0, 0.5, 'Actual')



```
[ ]: from tensorflow import keras
from tensorflow.keras import layers
#Defining Neural Network
model = Sequential()
#Non-trainable embedding layer
model.add(Embedding(max_features, output_dim=embed_size,
    ↳weights=[embedding_matrix], input_length=100, 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(learning_rate = 0.01),
    ↳loss='binary_crossentropy', metrics=['accuracy'])
```

```
[17]: # Model Analysis
print("Accuracy of the model on Training Data is - " , model.
      ↪evaluate(x_train,y_train)[1]*100 , "%")
print("Accuracy of the model on Testing Data is - " , model.
      ↪evaluate(X_test,y_test)[1]*100 , "%")
```

```
1053/1053 [=====] - 29s 28ms/step - loss: 0.2320 -
accuracy: 0.9086
```

```
Accuracy of the model on Training Data is - 90.85617661476135 %
```

```
351/351 [=====] - 14s 41ms/step - loss: 0.2291 -
accuracy: 0.9061
```

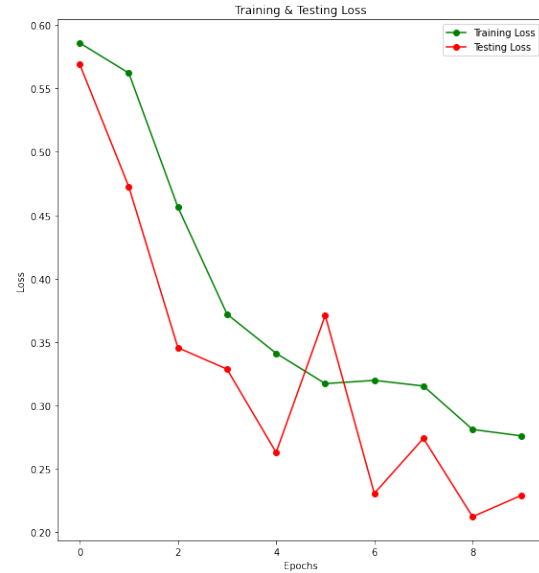
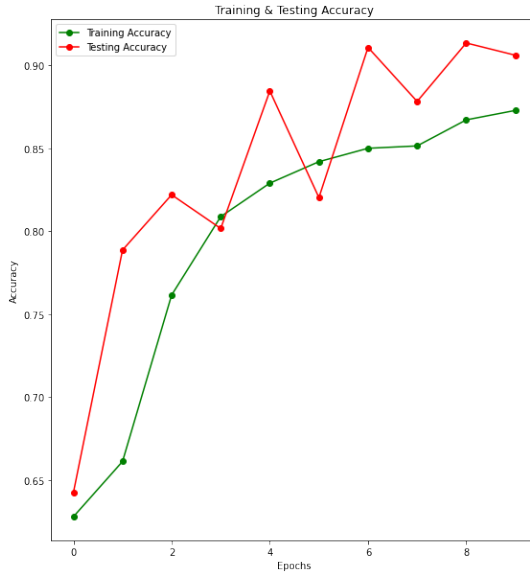
```
Accuracy of the model on Testing Data is - 90.61024785041809 %
```

```
[23]: epochs = [i for i in range(10)]
fig , ax = plt.subplots(1,2)
train_acc = history.history['accuracy']
train_loss = history.history['loss']
val_acc = history.history['val_accuracy']
val_loss = history.history['val_loss']

fig.set_size_inches(20,10)

ax[0].plot(epochs , train_acc , 'go-' , label = 'Training Accuracy')
ax[0].plot(epochs , val_acc , 'ro-' , label = 'Testing Accuracy')
ax[0].set_title('Training & Testing Accuracy')
ax[0].legend()
ax[0].set_xlabel("Epochs")
ax[0].set_ylabel("Accuracy")

ax[1].plot(epochs , train_loss , 'go-' , label = 'Training Loss')
ax[1].plot(epochs , val_loss , 'ro-' , label = 'Testing Loss')
ax[1].set_title('Training & Testing Loss')
ax[1].legend()
ax[1].set_xlabel("Epochs")
ax[1].set_ylabel("Loss")
plt.show()
```



```
[48]: pred = model.predict(X_test)
      pred[:5]
```

```
[48]: array([[5.1491559e-02],
             [7.5027813e-08],
             [6.7804095e-08],
             [1.5647142e-06],
             [9.8565006e-01]], dtype=float32)
```

```
[49]: pred = list(pred)
      test_result = []
      for i in range(len(pred)):
          test_result.append(pred[i][0])
```

```
[50]: confusion = pd.DataFrame({'Pred':test_result, 'Truth':list(y_test)})
      confusion['binary_pred'] = (confusion['Pred'] > 0.5).astype(int)
```

```
[63]: cm_DL = confusion_matrix(confusion['Truth'],confusion['binary_pred'])
      cm_DL
```

```
[63]: array([[5458,  400],
             [ 654, 4713]])
```

```
[64]: plt.figure(figsize = (10,8))
      sns.heatmap(cm_DL,cmap= "Blues", linecolor = 'black' , xticklabels = _
      ↪ ['Fake','Real'] , yticklabels = ['Fake','Real'])
      plt.xlabel("Predicted")
      plt.ylabel("Actual")
```

[64]: Text(69.0, 0.5, 'Actual')

