Fake_Real_News_Detective_Data_Visualization_LSTM_Train-Test

November 1, 2021

Data used for this project is from Kaggle, you can find it in the link below. For the project details, please see the report. Because the **Deep Learning** model need GPU, this document was run on the google colab, and the **Machine Learning** model is **not** include in this document.
Data souce Link

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

[ ]:
```
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
```

The cell above has been written as a class named *clean* for cleaning the data in further work. The commented out code *import clean* below is import this document.
All the ***.py*** file can be found in the submit zipped file, each member's code is in a separate folder.

[ ]:
```
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
```

[ ]:
```
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]
```

[ ]:
```
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']
```

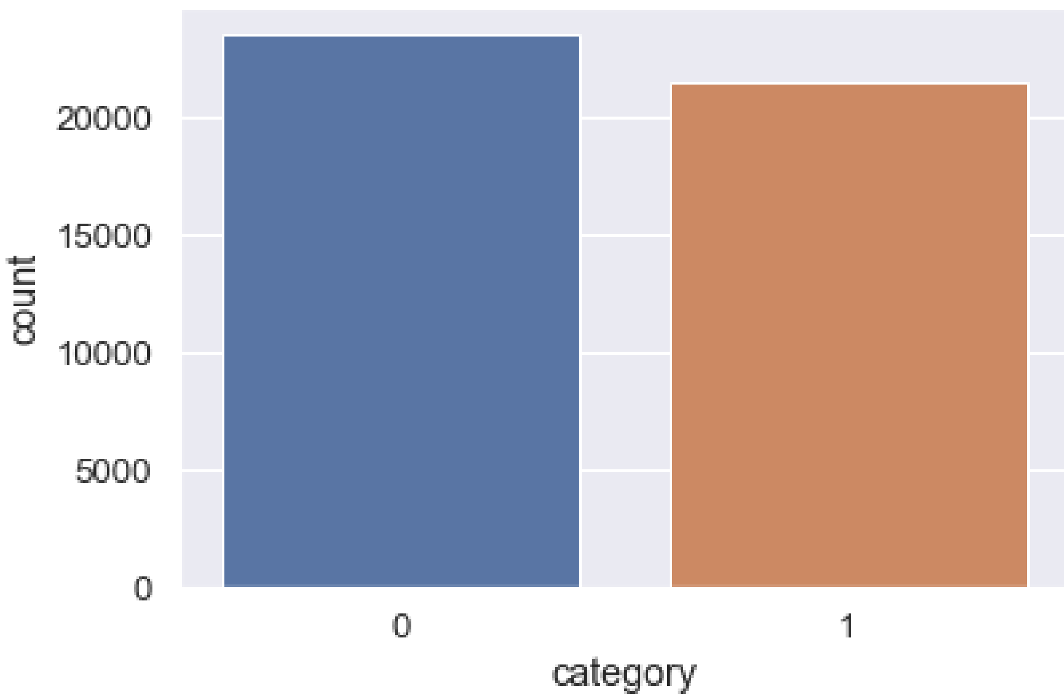
```
    df = pd.DataFrame({'text':clean_text,'category':df['category']})
    # df.head()
    # data_cleaning = clean()
    # df['text'] = df['text'].apply(data_cleaning.denoise_text)
    # WHEN I USE CLASS I'M UNABLE TO USE 'apply' FUNCTION, STILL NEED TO FIND␣
↪OUT WHY, USING LOOPS ARE SLOW
    print('============== 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])
```

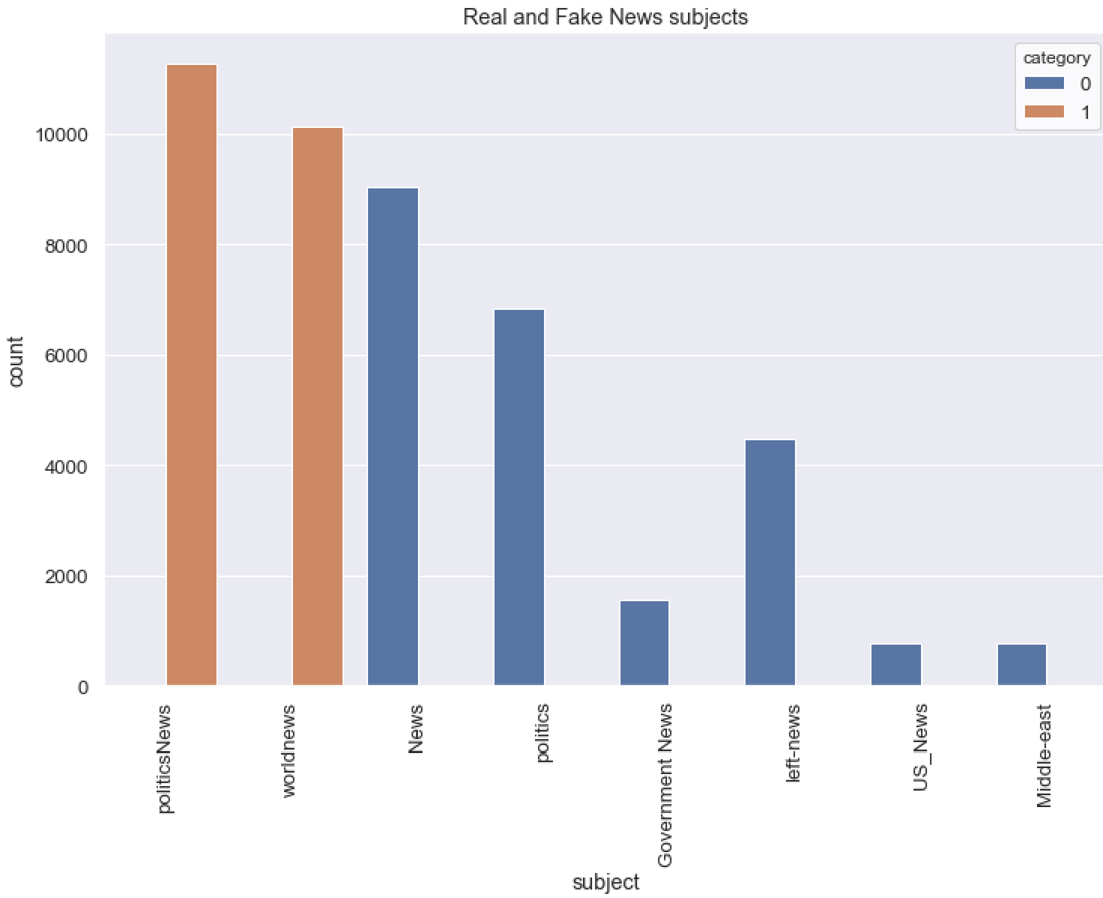
```
    sns.distplot(word.map(lambda x: np.mean(x)),ax=ax1,color='red')
    ax1.set_title('Original text')
    word=df[df['category']==0]['text'].str.split().apply(lambda x : [len(i) for␣
 ↪i in x])
    sns.distplot(word.map(lambda x: np.mean(x)),ax=ax2,color='green')
    ax2.set_title('Fake text')
    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()))
```

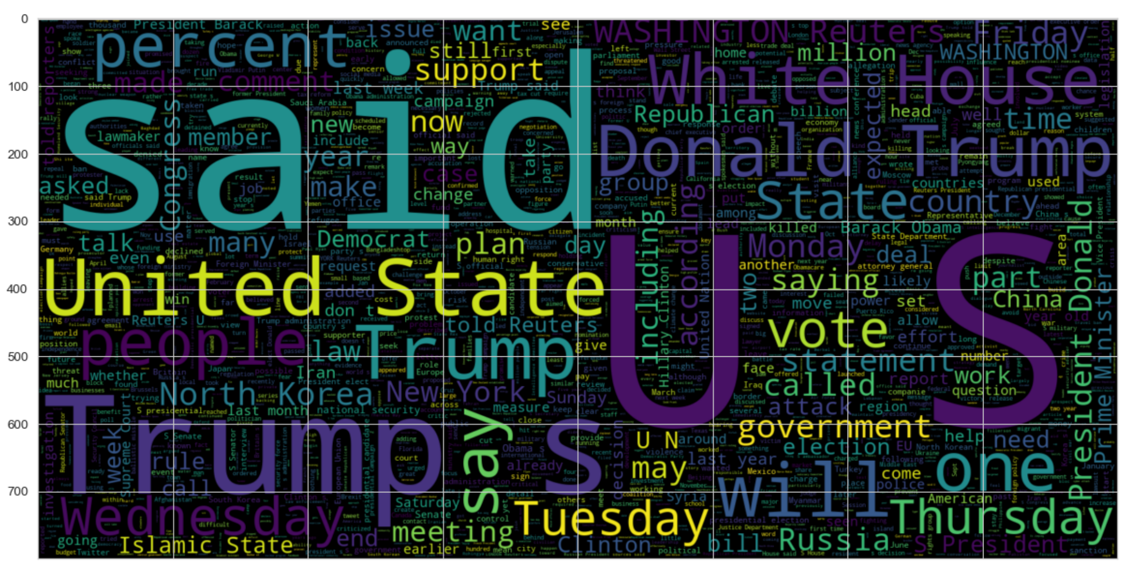
```
===================== Real Data Preview =====================
                                               title  \
0  As U.S. budget fight looms, Republicans flip t…
1  U.S. military to accept transgender recruits o…
2  Senior U.S. Republican senator: 'Let Mr. Muell…
3  FBI Russia probe helped by Australian diplomat…
4  Trump wants Postal Service to charge 'much mor…

                                                text       subject  \
0  WASHINGTON (Reuters) - The head of a conservat…  politicsNews
1  WASHINGTON (Reuters) - Transgender people will…  politicsNews
2  WASHINGTON (Reuters) - The special counsel inv…  politicsNews
3  WASHINGTON (Reuters) - Trump campaign adviser …  politicsNews
```

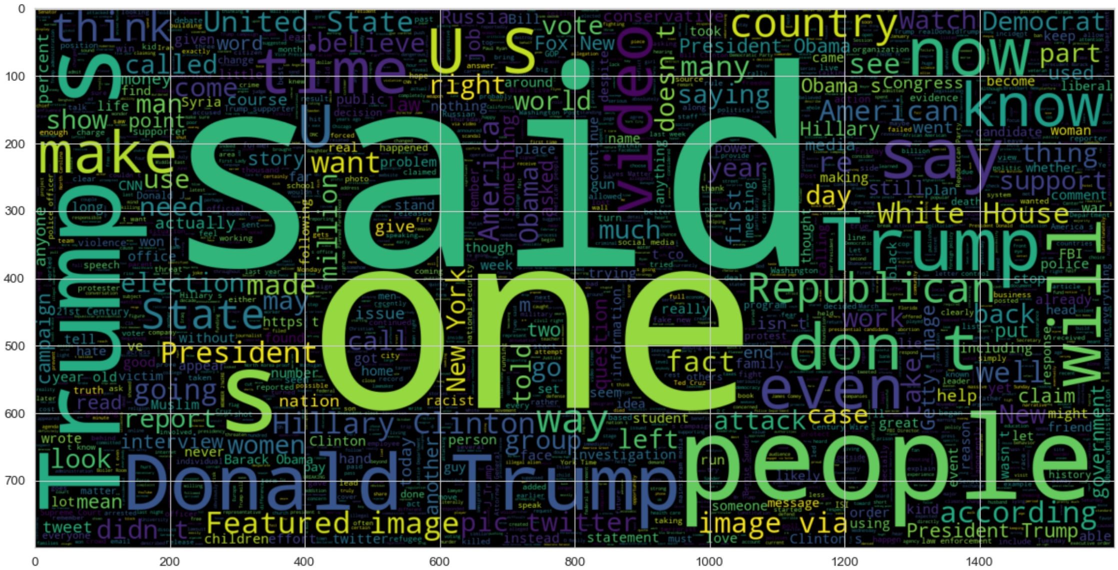
```
4  SEATTLE/WASHINGTON (Reuters) - President Donal…  politicsNews

                 date
0  December 31, 2017
1  December 29, 2017
2  December 31, 2017
3  December 30, 2017
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
```

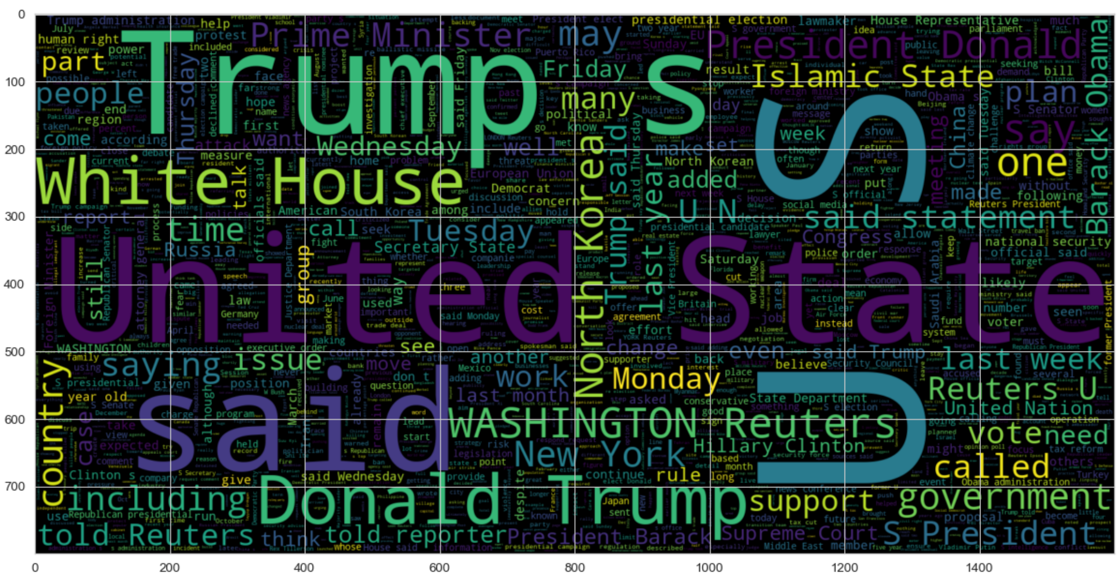
```
US_News             783
Middle-east         778
Name: subject, dtype: int64
========== Comparie the Number of Real and Fake News =========
Check whether the data set has null values
 title       0
text        0
subject     0
date        0
category    0
dtype: int64
Check news subjects
 politicsNews       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 =============
```

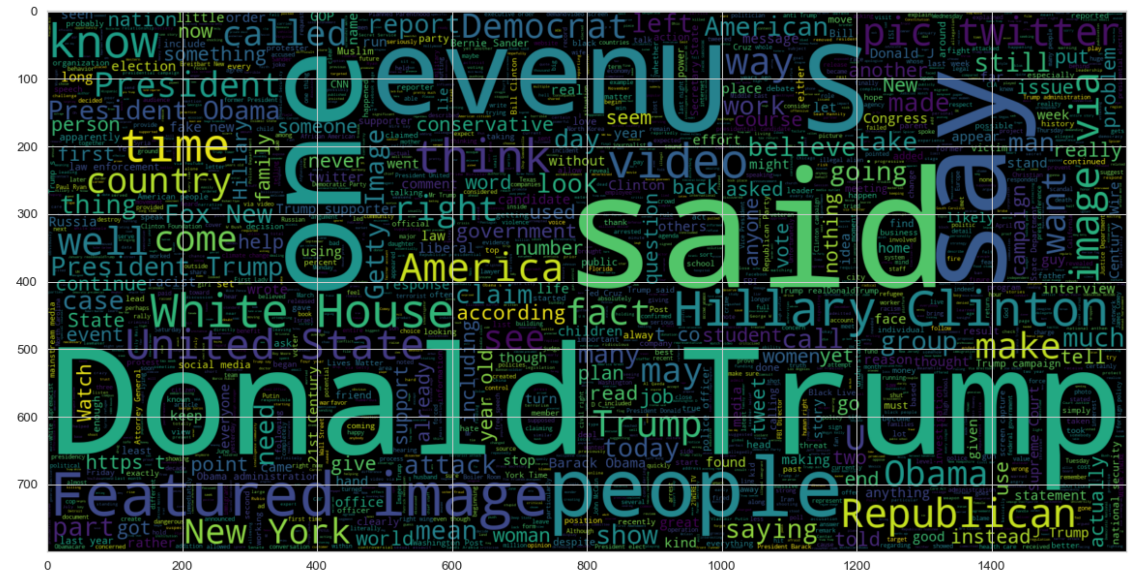


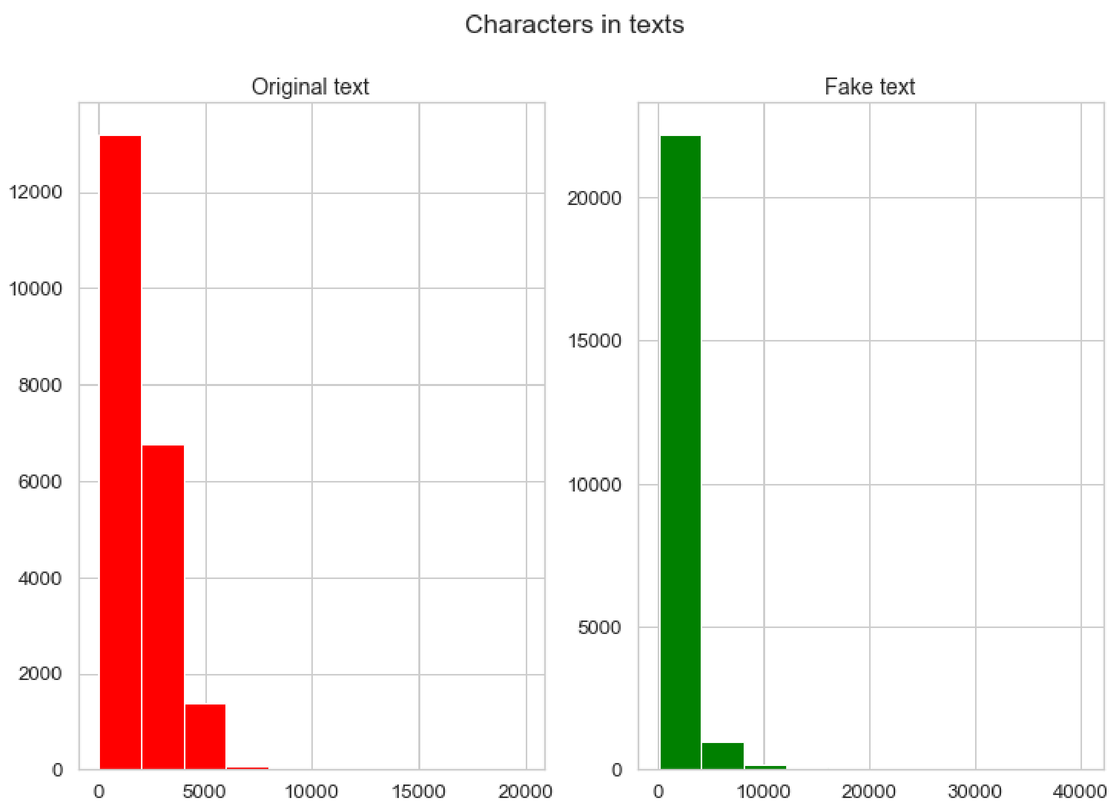

Real and Fake News subjects
category
0
1
count
10000
8000
6000
4000
2000
0
politicsNews
worldnews
News
politics
Government News
left-news
US_News
Middle-east
subject


said
US
Trump
United State
White House
Donald Trump
percent
people
Wednesday
Tuesday
Thursday
vote
one
will
say
North Korea
New York
government
Islamic State
Russia
Clinton
Democrat
Republican
support
Reuters
WASHINGTON
Friday
Monday
Barack Obama
China
President Donald
Prime Minister

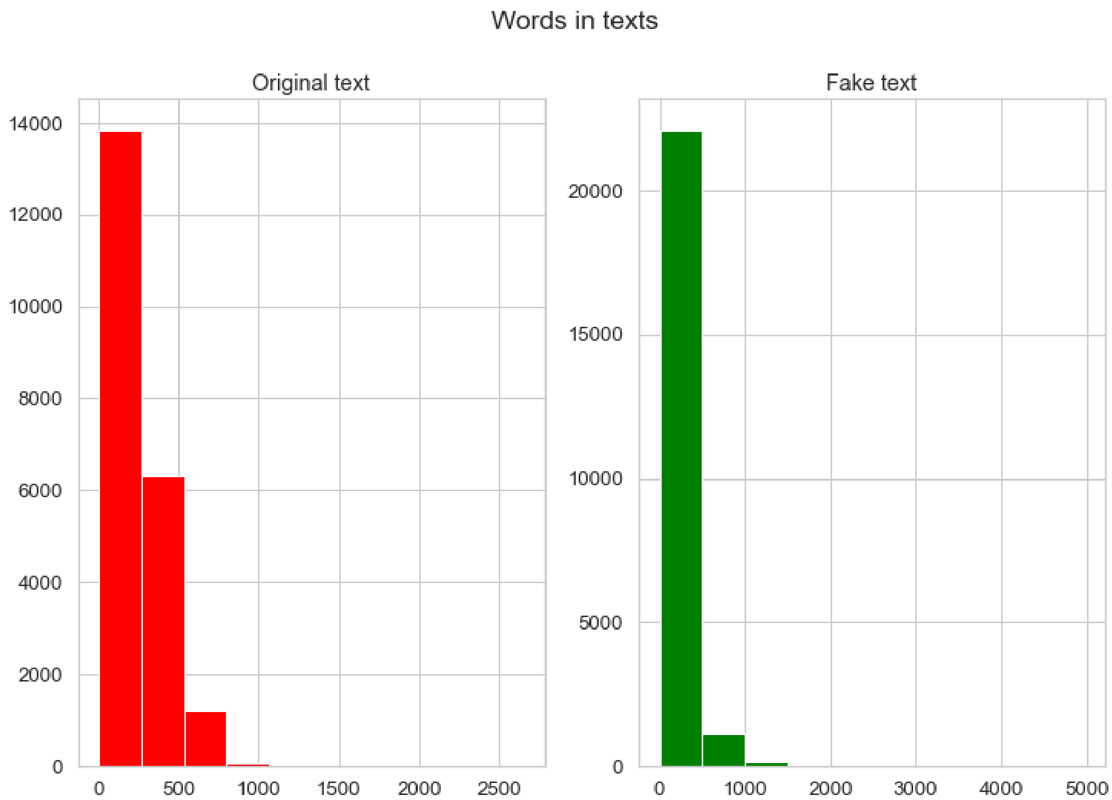

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200
1400
said
one
people
Trump s
Donald Trump
Trump
know
now
time
U S
video
will
even
don t
say
Republican
make
country
United State
White House
Hillary Clinton
Featured image
image via
according
President
State
think


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200
1400
United State
Trump s
U S
said
White House
Donald Trump
WASHINGTON Reuters
New York
North Korea
Prime Minister may
President Donald
Islamic State
government
including
told Reuters
Monday
vote
called
one
Barack Obama
said statement
last week

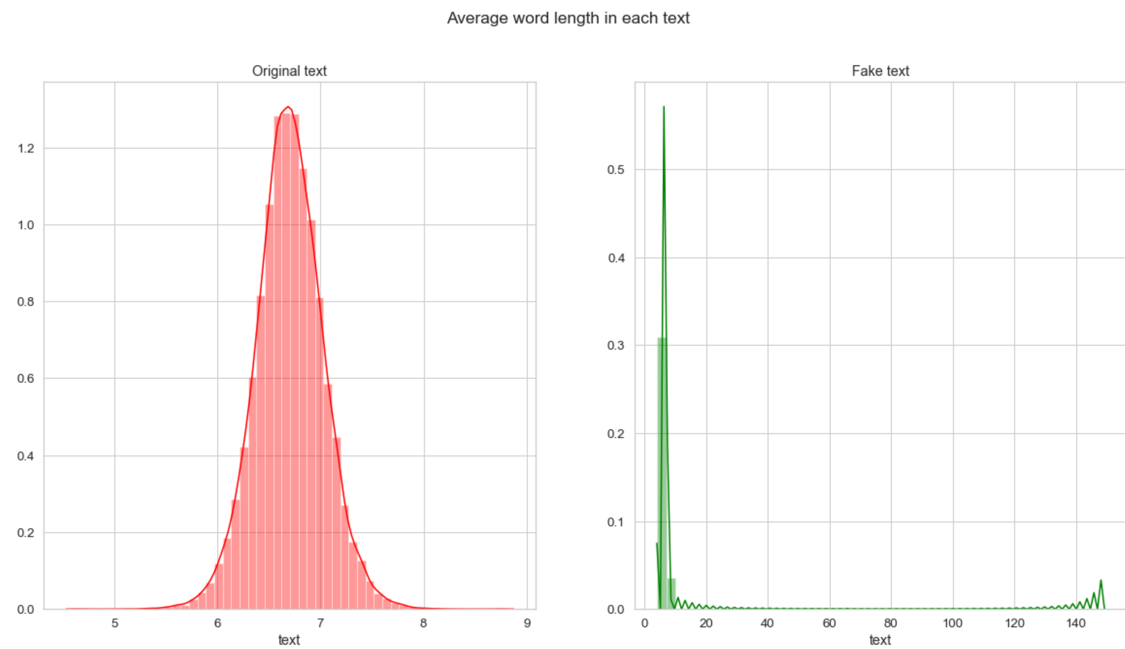


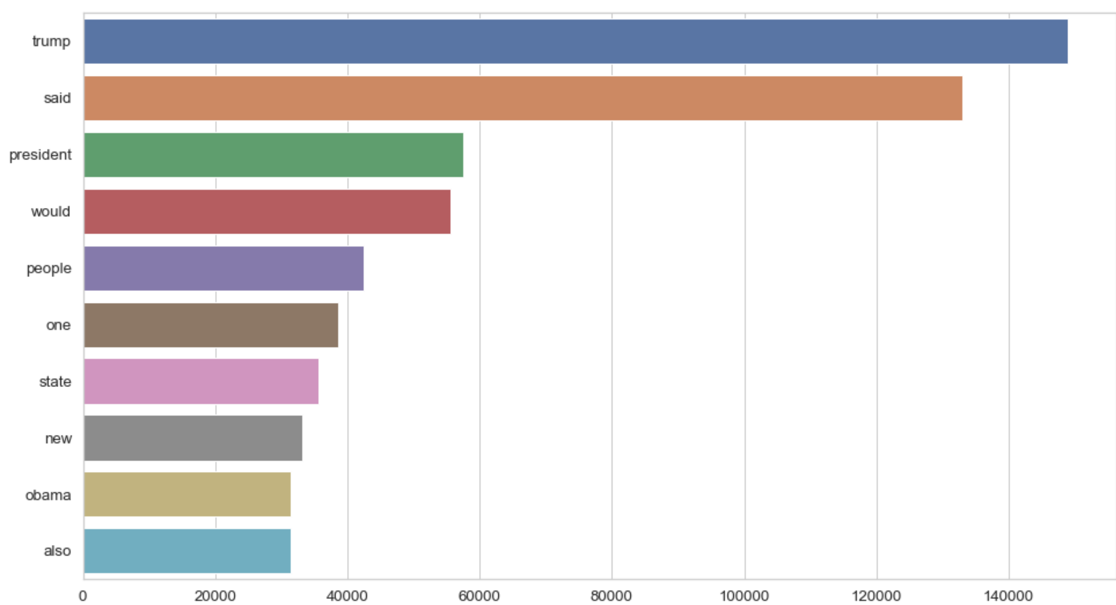


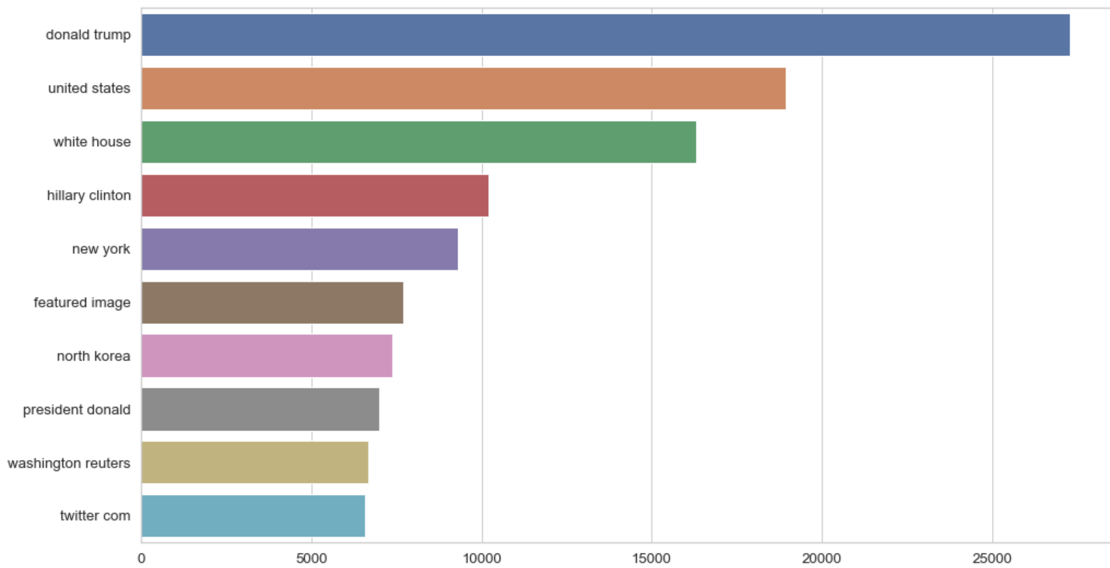

```
The distribution of both seems to be a bit different.
 2500 characters in text is the most common in original text category
 while around 5000 characters in text are most common in fake text category.
```

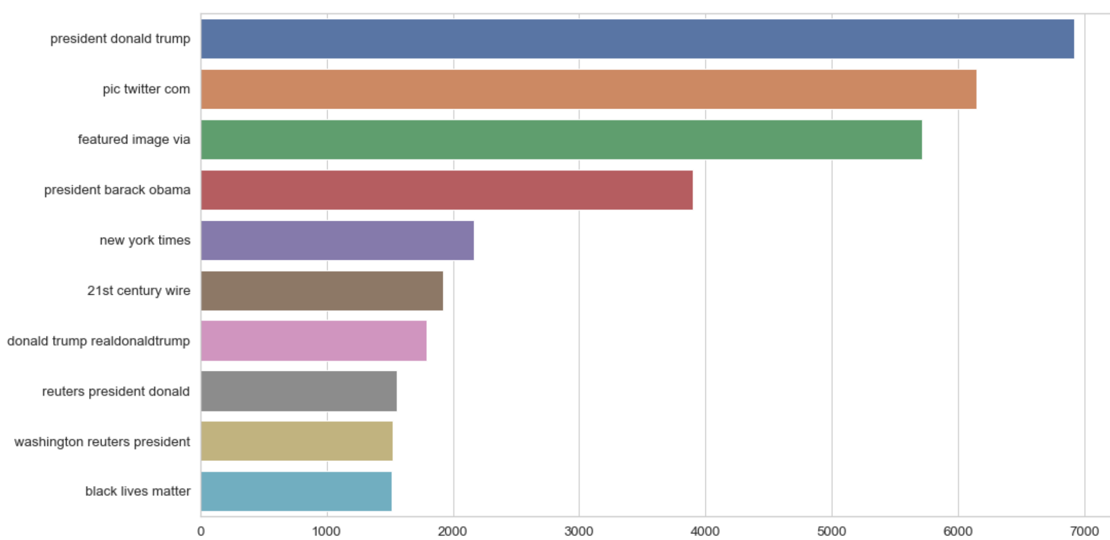



```
Top 5 Words
 ['WASHINGTON', '(Reuters)', 'head', 'conservative', 'Republican']
Numbers of most common words {'Trump': 111503, 'said': 93162, 'would': 54613,
'U.S.': 50441, 'President': 33180, 'people': 33115, 'also': 30325, 'one': 29370,
'Donald': 27795, 'said.': 26190}
```


Average word length in each text
Original text
Fake text
text
text


trump
said
president
would
people
one
state
new
obama
also





[ ]:
```
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
```

```python
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
```

[ ]:
```python
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']})
```

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

In order to adjust the model parameters conveniently, we save the cleaned data in the file *combine.csv*.

[ ]:
```python
# for local use
```

```
df = pd.read_csv('/Users/yuechenjiang/Desktop/project660/code-data/combined.
↪csv')
```

[ ]:
```
# only use for colab
tf.test.gpu_device_name()
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))

# read file from cloud file
import os
from google.colab import drive
drive.mount('/content/drive')

path = "/content/drive/My Drive/Colab Notebooks/code&data"

os.chdir(path)
os.listdir(path)

df = pd.read_csv('combined.csv')
```

```
Num GPUs Available:  1
Mounted at /content/drive
```

[ ]:
```
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 -> Repsesenting each word by a number
# Mapping of orginal word to number is preserved in word_index property of␣
↪tokenizer
# Tokenized applies basic processing like changing it to lower case,␣
↪explicitely 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)
```

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

[ ]:
```
EMBEDDING_FILE = 'train.txt'
```

[ ]:
```
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))
```

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

```
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2822:
FutureWarning: arrays to stack must be passed as a "sequence" type such as list
or tuple. Support for non-sequence iterables such as generators is deprecated as
of NumPy 1.16 and will raise an error in the future.
  if self.run_code(code, result):
```

[ ]:
```
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
```

[ ]:
```
embed_size
```

[ ]: 100

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

[ ]: (10000, 100)

[ ]:
```
emb_mean
```

[ ]: 1615.2051

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

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

[ ]:
```
from tensorflow import keras
from tensorflow.keras import layers
#Defining Neural Network
```

```
model = Sequential()
#Non-trainable embeddidng 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 ,dropout = 0.25))
model.add(LSTM(units=64 , 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'])
```

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

```
Model: "sequential"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (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

_________________________________________________________________
```

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

```
Epoch 1/30
WARNING:tensorflow:Model was constructed with shape (None, 300) for input
KerasTensor(type_spec=TensorSpec(shape=(None, 300), dtype=tf.float32,
name='embedding_input'), name='embedding_input', description="created by layer
'embedding_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_input'), name='embedding_input', description="created by layer
'embedding_input'"), but it was called on an input with incompatible shape
(None, 101).
```

```
132/132 [==============================] - ETA: 0s - loss: 0.5908 - accuracy:
0.6350WARNING:tensorflow:Model was constructed with shape (None, 300) for input
KerasTensor(type_spec=TensorSpec(shape=(None, 300), dtype=tf.float32,
name='embedding_input'), name='embedding_input', description="created by layer
'embedding_input'"), but it was called on an input with incompatible shape
(None, 101).
132/132 [==============================] - 147s 1s/step - loss: 0.5908 -
accuracy: 0.6350 - val_loss: 0.5470 - val_accuracy: 0.6490
Epoch 2/30
132/132 [==============================] - 139s 1s/step - loss: 0.5754 -
accuracy: 0.6448 - val_loss: 0.5722 - val_accuracy: 0.6420
Epoch 3/30
132/132 [==============================] - 140s 1s/step - loss: 0.5791 -
accuracy: 0.6359 - val_loss: 0.5641 - val_accuracy: 0.6440

Epoch 00003: ReduceLROnPlateau reducing learning rate to 0.004999999888241291.
Epoch 4/30
132/132 [==============================] - 140s 1s/step - loss: 0.5497 -
accuracy: 0.6825 - val_loss: 0.5507 - val_accuracy: 0.6927
Epoch 5/30
132/132 [==============================] - 140s 1s/step - loss: 0.5261 -
accuracy: 0.7090 - val_loss: 0.5174 - val_accuracy: 0.7231
Epoch 6/30
132/132 [==============================] - 141s 1s/step - loss: 0.4300 -
accuracy: 0.7910 - val_loss: 0.3133 - val_accuracy: 0.8615
Epoch 7/30
132/132 [==============================] - 139s 1s/step - loss: 0.3804 -
accuracy: 0.8186 - val_loss: 0.3730 - val_accuracy: 0.8293
Epoch 8/30
132/132 [==============================] - 139s 1s/step - loss: 0.3623 -
accuracy: 0.8325 - val_loss: 0.3575 - val_accuracy: 0.8364

Epoch 00008: ReduceLROnPlateau reducing learning rate to 0.0024999999441206455.
Epoch 9/30
132/132 [==============================] - 141s 1s/step - loss: 0.3287 -
accuracy: 0.8508 - val_loss: 0.2662 - val_accuracy: 0.8800
Epoch 10/30
132/132 [==============================] - 144s 1s/step - loss: 0.2998 -
accuracy: 0.8646 - val_loss: 0.2358 - val_accuracy: 0.8911
Epoch 11/30
132/132 [==============================] - 144s 1s/step - loss: 0.2734 -
accuracy: 0.8799 - val_loss: 0.2247 - val_accuracy: 0.9022
Epoch 12/30
132/132 [==============================] - 142s 1s/step - loss: 0.2629 -
accuracy: 0.8843 - val_loss: 0.2124 - val_accuracy: 0.9089
Epoch 13/30
132/132 [==============================] - 144s 1s/step - loss: 0.2488 -
accuracy: 0.8931 - val_loss: 0.1926 - val_accuracy: 0.9156
```

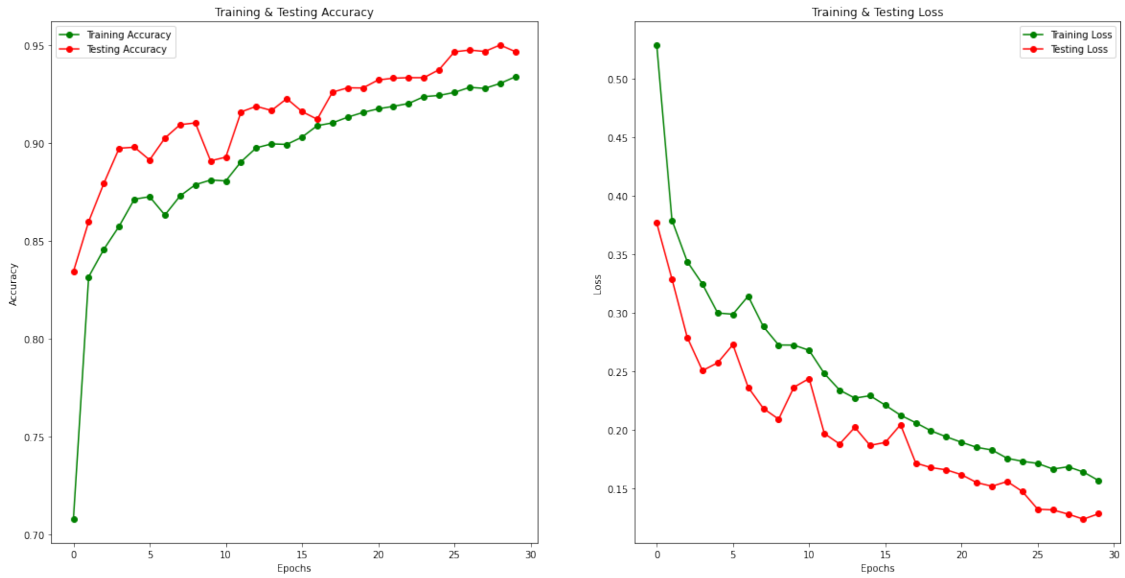
```
Epoch 14/30
132/132 [==============================] - 143s 1s/step - loss: 0.2302 -
accuracy: 0.8990 - val_loss: 0.1943 - val_accuracy: 0.9154
Epoch 15/30
132/132 [==============================] - 144s 1s/step - loss: 0.2253 -
accuracy: 0.9027 - val_loss: 0.2087 - val_accuracy: 0.9053

Epoch 00015: ReduceLROnPlateau reducing learning rate to 0.0012499999720603228.
Epoch 16/30
132/132 [==============================] - 143s 1s/step - loss: 0.2092 -
accuracy: 0.9096 - val_loss: 0.1862 - val_accuracy: 0.9163
Epoch 17/30
132/132 [==============================] - 140s 1s/step - loss: 0.2075 -
accuracy: 0.9115 - val_loss: 0.1720 - val_accuracy: 0.9267
Epoch 18/30
132/132 [==============================] - 140s 1s/step - loss: 0.2032 -
accuracy: 0.9124 - val_loss: 0.1651 - val_accuracy: 0.9291
Epoch 19/30
132/132 [==============================] - 139s 1s/step - loss: 0.2001 -
accuracy: 0.9146 - val_loss: 0.1506 - val_accuracy: 0.9347
Epoch 20/30
132/132 [==============================] - 139s 1s/step - loss: 0.1944 -
accuracy: 0.9183 - val_loss: 0.1559 - val_accuracy: 0.9335
Epoch 21/30
132/132 [==============================] - 139s 1s/step - loss: 0.1937 -
accuracy: 0.9180 - val_loss: 0.1457 - val_accuracy: 0.9386
Epoch 22/30
132/132 [==============================] - 139s 1s/step - loss: 0.1855 -
accuracy: 0.9216 - val_loss: 0.1512 - val_accuracy: 0.9363
Epoch 23/30
132/132 [==============================] - 139s 1s/step - loss: 0.1851 -
accuracy: 0.9219 - val_loss: 0.1275 - val_accuracy: 0.9490
Epoch 24/30
132/132 [==============================] - 139s 1s/step - loss: 0.1851 -
accuracy: 0.9220 - val_loss: 0.1433 - val_accuracy: 0.9408
Epoch 25/30
132/132 [==============================] - 139s 1s/step - loss: 0.1817 -
accuracy: 0.9228 - val_loss: 0.1464 - val_accuracy: 0.9380

Epoch 00025: ReduceLROnPlateau reducing learning rate to 0.0006249999860301614.
Epoch 26/30
 87/132 [==================>…] - ETA: 42s - loss: 0.1751 - accuracy:
0.9265
```

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

```python
print("Accuracy of the model on Testing Data is - " , model.
 ↪evaluate(X_test,y_test)[1]*100 , "%")
```

```
1053/1053 [==============================] - 19s 18ms/step - loss: 0.1258 -
accuracy: 0.9455
Accuracy of the model on Training Data is -  94.55053210258484 %
351/351 [==============================] - 6s 18ms/step - loss: 0.1283 -
accuracy: 0.9465
Accuracy of the model on Testing Data is -  94.64588165283203 %
```

[ ]:
```python
epochs = [i for i in range(30)]
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()
```



[ ]:
```python
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_input'), name='embedding_input', description="created by layer
'embedding_input'"), but it was called on an input with incompatible shape
(None, 101).
```

[ ]:
```
array([[2.2828460e-01],
       [1.5324001e-06],
       [3.8175322e-06],
       [1.0576205e-04],
       [9.9944443e-01]], dtype=float32)
```

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

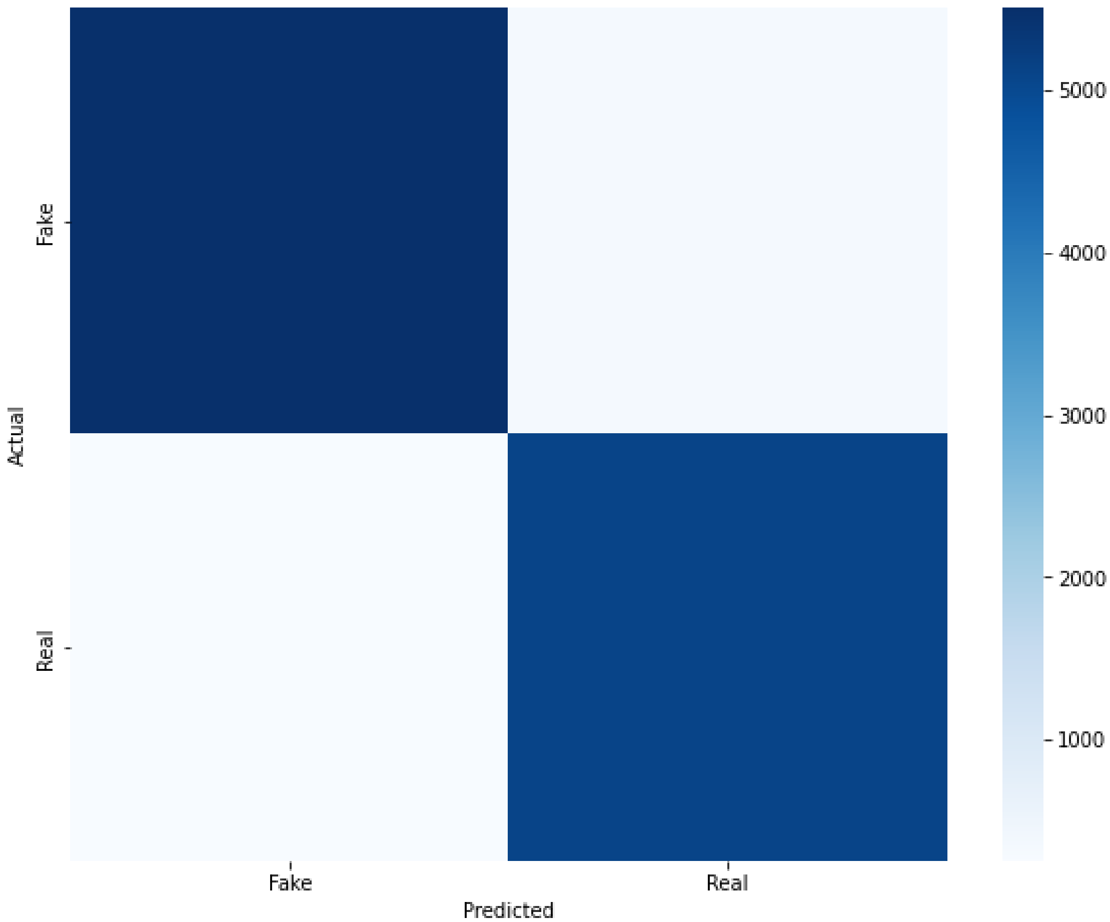
[ ]:
```python
confusion = pd.DataFrame({'Pred':test_result, 'Truth':list(y_test)})
confusion['binary_pred'] = (confusion['Pred'] > 0.5).astype(int)
```

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

[ ]:
```
array([[5511,  347],
       [ 254, 5113]])
```

[ ]:
```
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")
```

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



We split the original dataset as two part for *training* and *testing*, and we got an accuracy of 94.55% on the *training* data and 94.65% for the *testing* data. Now we use different data souce on Kaggle and test the model again. Visualization of the data can be found in the **machine learning** model part and midterm report.
data source Link

[ ]:
```
df_test = pd.read_csv('fake_or_real_news.csv')
```

[ ]:
```
df_test['text'] = df_test['text'] + " " + df_test['title']
```

```python
import clean
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
clean_text = []
for i in df['text']:
    data_cleaning = clean.clean(text = i)
    clean_text.append(data_cleaning.denoise_text(i))
del df['text']

label = []
for i in df_test['label']:
    if i == 'FAKE':
        label.append(0)
    elif i == 'REAL':
        label.append(1)
    else:
        label.append2

df = pd.DataFrame({'text':clean_text,'label':label})
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
```

[ ]:
```python
x_test2 = df['text']
y_test2 = df['label']
```

[ ]:
```python
max_features = 10000
maxlen = 101
```

[ ]:
```python
tokenizer = text.Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(x_test2)
tokenized_test2 = tokenizer.texts_to_sequences(x_test2)
x_test2 = sequence.pad_sequences(tokenized_test2, maxlen=maxlen)
```

[ ]:
```python
pred = model.predict(x_test2)
pred[:5]
```

[ ]:
```
array([[0.36004308],
       [0.23542677],
       [0.78353333],
       [0.9822192 ],
       [0.8281593 ]], dtype=float32)
```

[ ]:
```python
pred = list(pred)
test_result = []
for i in range(len(pred)):
```

```
    test_result.append(pred[i][0])
```

[ ]:
```
confusion = pd.DataFrame({'Pred':test_result, 'Truth':list(y_test2)})
confusion['binary_pred'] = (confusion['Pred'] > 0.5).astype(int)
```

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

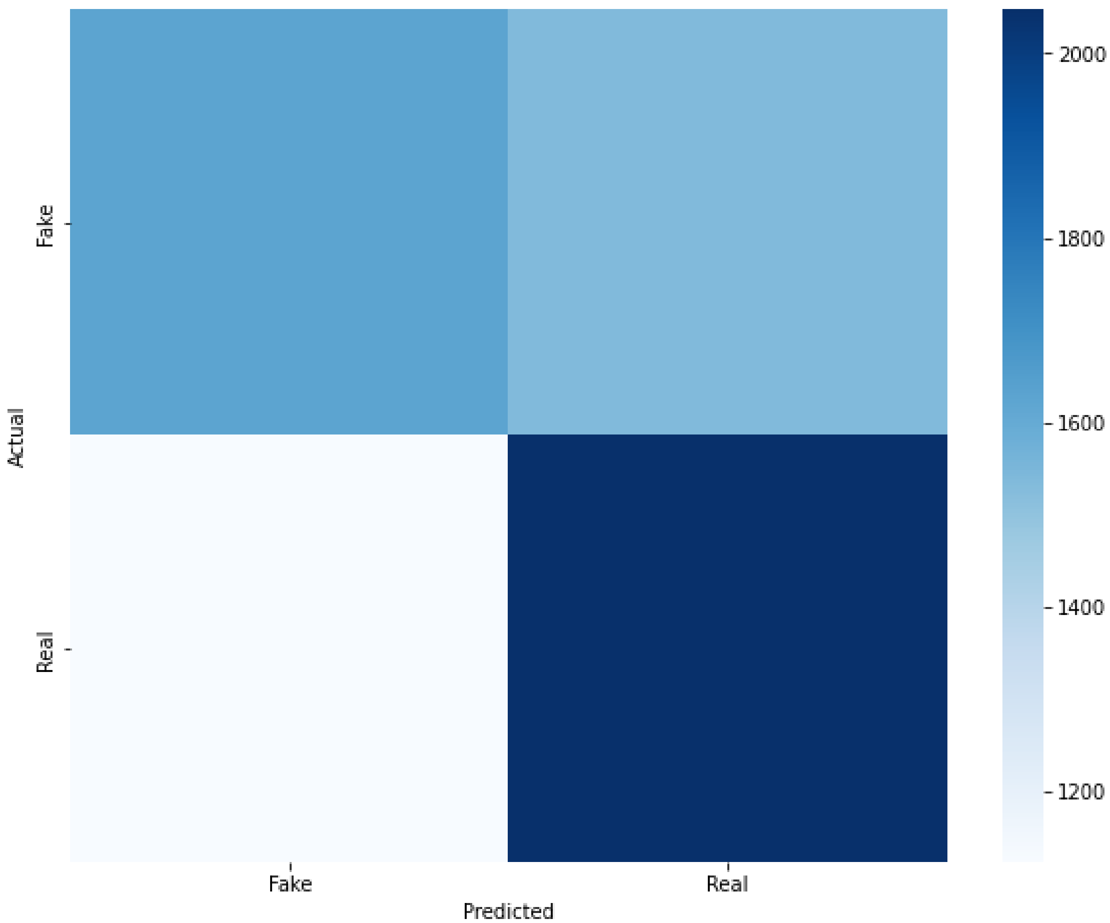
[ ]:
```
array([[1626, 1538],
       [1123, 2048]])
```

[ ]:
```
print("Accuracy of the model on Other Testing Data is - " , model.
 ↪evaluate(x_test2,y_test2)[1]*100 , "%")
```

```
198/198 [==============================] - 4s 18ms/step - loss: 1.4007 -
accuracy: 0.5800
Accuracy of the model on Other Testing Data is -  57.99526572227478 %
```

[ ]:
```
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")
```

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



As the result above, we can see the model need to be improved in further work.

This document is written by team member Yuechen Jiang