Fake_Real_News_Detectivs_Data_Visualization-Machine_Learning_Model_Train-Test

November 3, 2021

[13]:
```
import numpy as np
import pandas as pd
import seaborn as sns
from collections import Counter
from wordcloud import WordCloud,STOPWORDS
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import␣
↪classification_report,confusion_matrix,accuracy_score
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
```

[2]:
```
import clean
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']})
```

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

In this part we use unigram and bigram with max fatures of 15000 to transfer the text into matrix, these two parameters need to be adjusted in the futher work in order to get better result. Because running the models are time consuming, we didn't do it yet.

[8]:
```
df = pd.read_csv('combined.csv')
X = df['text']
Y = df['category']
x_train, x_test, y_train, y_test = model_selection.train_test_split(X, Y ,␣
 ↪train_size=0.8, shuffle = True, test_size=0.2, random_state=1)
tfidf_vectorizer = TfidfVectorizer(ngram_range=(1,2), max_features=15000,␣
 ↪use_idf=True, stop_words='english')
x_train = tfidf_vectorizer.fit_transform(x_train)
x_test = tfidf_vectorizer.fit_transform(x_test)
```

[9]:
```
print('train:',x_train.shape,'\n','test:',x_test.shape)
```

```
train: (35918, 15000)
 test: (8980, 15000)
```

[23]:
```
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',
                'NaiveBayesClassifier']
```

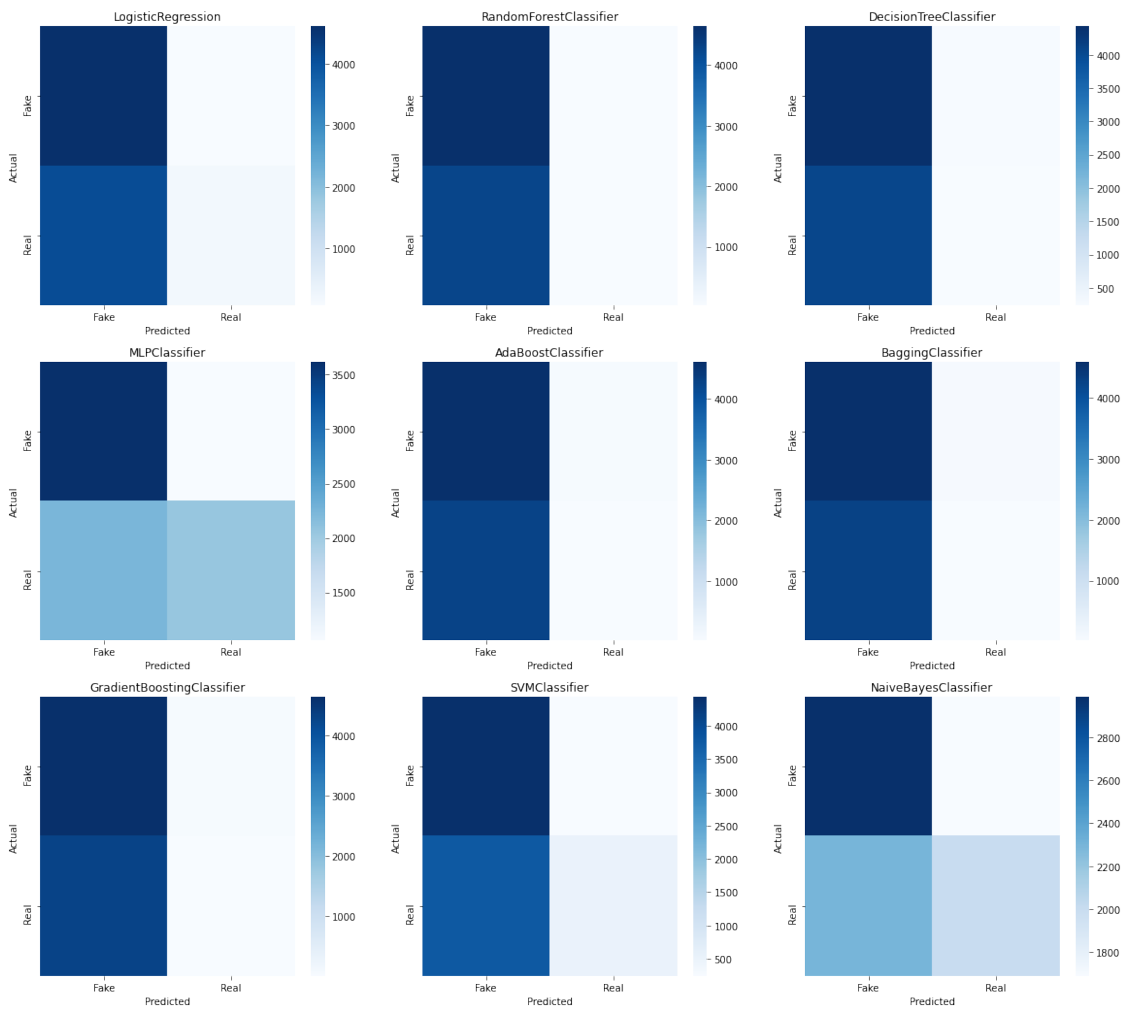
[24]:
```
acc = []
cms = []
for model in models:
    model.fit(x_train,y_train)
    # model_acc = model.score(x_test, y_test)*100
    acc.append(model.score(x_test, y_test))
    y_pred = model.predict(x_test)
    cm = confusion_matrix(y_test,y_p
                          red)
    cms.append(cm)
    print(model,'\n',cm)

from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(x_train.toarray(),y_train)
acc.append(model.score(x_test.toarray(), y_test))
y_pred = model.predict(x_test.toarray())
cm = confusion_matrix(y_test,y_pred)
cms.append(cm)
print('GaussianNB()\n',cm)
```

```
LogisticRegression()
 [[4611   67]
 [4122  180]]
RandomForestClassifier()
 [[4638   40]
 [4271   31]]
DecisionTreeClassifier()
 [[4427  251]
 [4068  234]]
MLPClassifier(max_iter=100)
 [[3623 1055]
 [2240 2062]]
AdaBoostClassifier()
 [[4621   57]
 [4292   10]]
BaggingClassifier()
 [[4596   82]
 [4288   14]]
GradientBoostingClassifier()
 [[4639   39]
 [4301    1]]
SVC(kernel='linear')
 [[4439  239]
```

```
 [3776  526]]
GaussianNB()
 [[2990 1688]
 [2299 2003]]
```

[25]:

```python
fig,ax=plt.subplots(3,3,figsize=(20,18))
for i in range(len(cms)):
    plt.subplot(3, 3, 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])
```



[26]:

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

[26]:

```
                             name       acc
0              LogisticRegression  0.533519
1          RandomForestClassifier  0.519933
2          DecisionTreeClassifier  0.519042
3                   MLPClassifier  0.633073
4              AdaBoostClassifier  0.515702
5               BaggingClassifier  0.513363
6      GradientBoostingClassifier  0.516704
7                   SVMClassifier  0.552895
8           NaiveBayesClassifier  0.556013
```

When test the data with the data split from the orignal dataset, the best model is **MLP Classifier** with accuracy **63.31%**. But this is still **not good** enough. In futher work, first we need update the stop word dictionary, such as 'Trump','Donald Trump' are apprea in most files, it has no relationship with the news is real or fake. And for some word only high frenquency appreas in fake or real news need add more weight. And the **SVM** and **Naive Bayes** can be consider in futher work to.

Now we use different data souce on Kaggle and test the model again.

Data Visualization of Testing Data

[3]:

```python
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]
```

[14]:

```python
if __name__ == "__main__":
    # load the real news data and preview
    df_test = pd.read_csv('fake_or_real_news.csv')
    print('====================== Test Data Preview ======================')
    print(df_test.head())
    print('====================== Test Data Describe =====================')
    df_test.describe()
    print('Test Dataset Shape:','\n',df_test.shape)
    print('Test Columns name:','\n',df_test.columns)
```

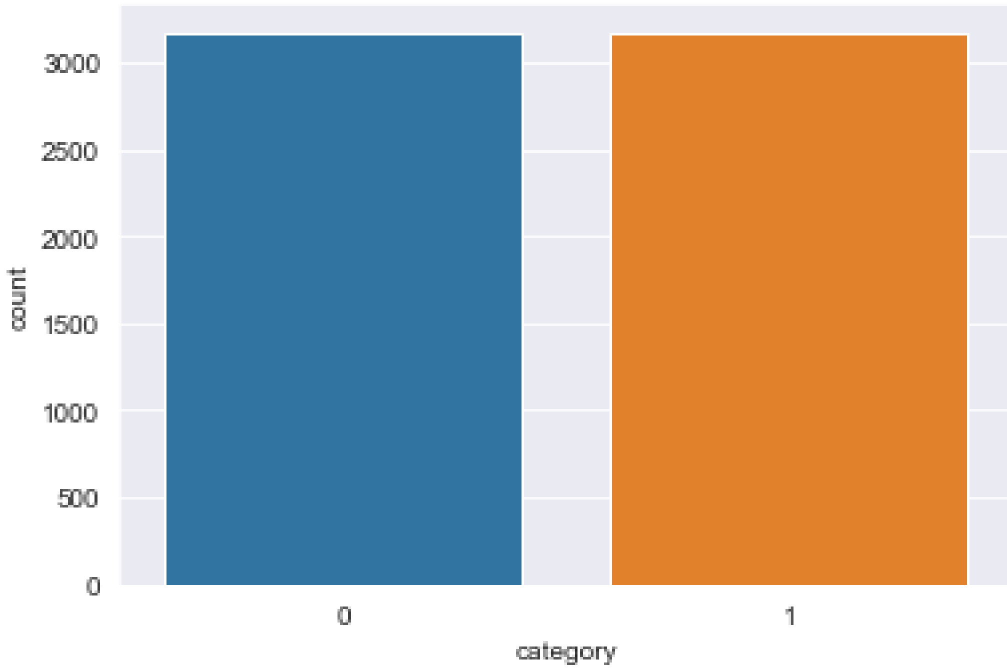
```
# Data visualization
df_test['text'] = df_test['text'] + " " + df_test['title']
label = []
for i in df_test['label']:
    if i == 'FAKE':
        label.append(0)
    elif i == 'REAL':
        label.append(1)
    else:
        label.append(2)

clean_text = []
for i in df_test['text']:
    data_cleaning = clean.clean(text = i)
    clean_text.append(data_cleaning.denoise_text(i))
del df_test['text']

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

print('=========== Comparie the Number of Real and Fake News ===========')
sns.set_style("darkgrid")
sns.countplot(df.category)

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
```

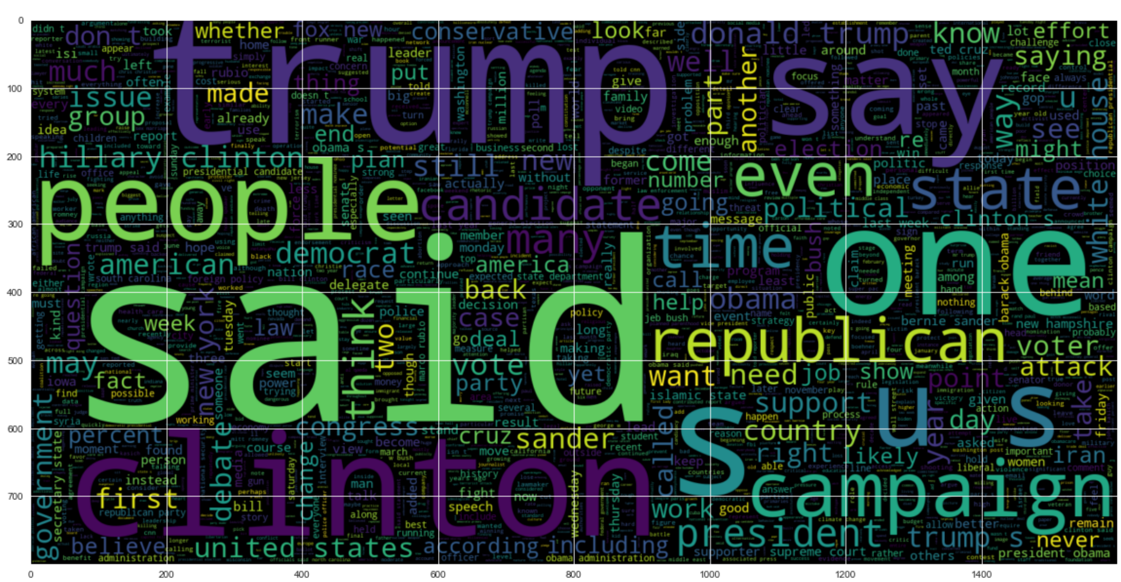
```
    wc = WordCloud(max_words = 2000 , width = 1600 , height = 800 , stopwords =␣
↪STOPWORDS).generate(" ".join(df[df.category == 1].text))
    plt.imshow(wc , interpolation = 'bilinear')
    print('============== Fake News World Cloud After Cleaning =============')
    plt.figure(figsize = (20,20)) # 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')
```

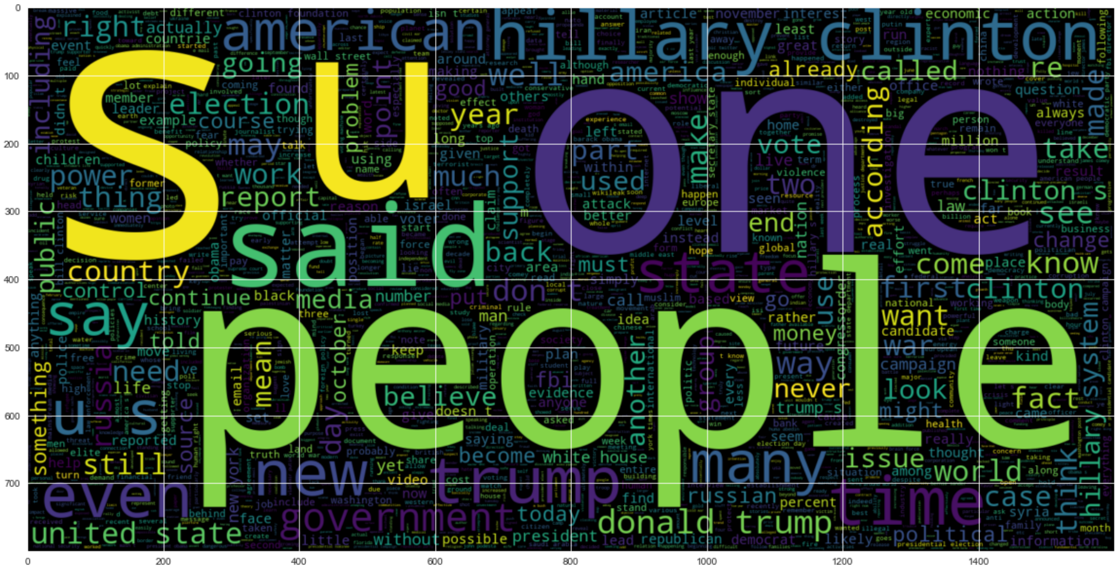
```python
corpus = get_corpus(df.text)
print('Top 5 Words','\n',corpus[:5])

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()))
```

```
====================== Test Data Preview ======================
   Unnamed: 0                                              title  \
0        8476                       You Can Smell Hillary’s Fear
1       10294  Watch The Exact Moment Paul Ryan Committed Pol…
2        3608        Kerry to go to Paris in gesture of sympathy
3       10142  Bernie supporters on Twitter erupt in anger ag…
4         875   The Battle of New York: Why This Primary Matters

                                                text label
0  Daniel Greenfield, a Shillman Journalism Fello…  FAKE
1  Google Pinterest Digg Linkedin Reddit Stumbleu…  FAKE
2  U.S. Secretary of State John F. Kerry said Mon…  REAL
3  — Kaydee King (@KaydeeKing) November 9, 2016 T…  FAKE
4  It's primary day in New York and front-runners…  REAL
====================== Test Data Describe =====================
Test Dataset Shape:
 (6335, 4)
Test Columns name:
 Index(['Unnamed: 0', 'title', 'text', 'label'], dtype='object')
=========== Comparie the Number of Real and Fake News ==========
```

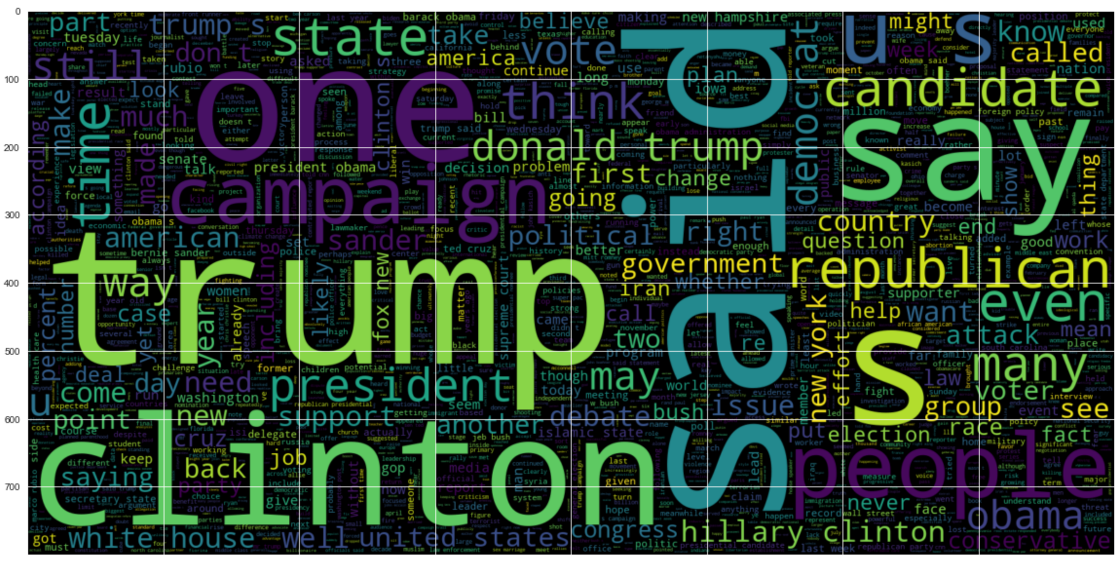
```
============= Real News World Cloud Before Cleaning =============
============= Fake News World Cloud Before Cleaning =============
============== Real News World Cloud After Cleaning =============
============== Fake News World Cloud After Cleaning =============
```



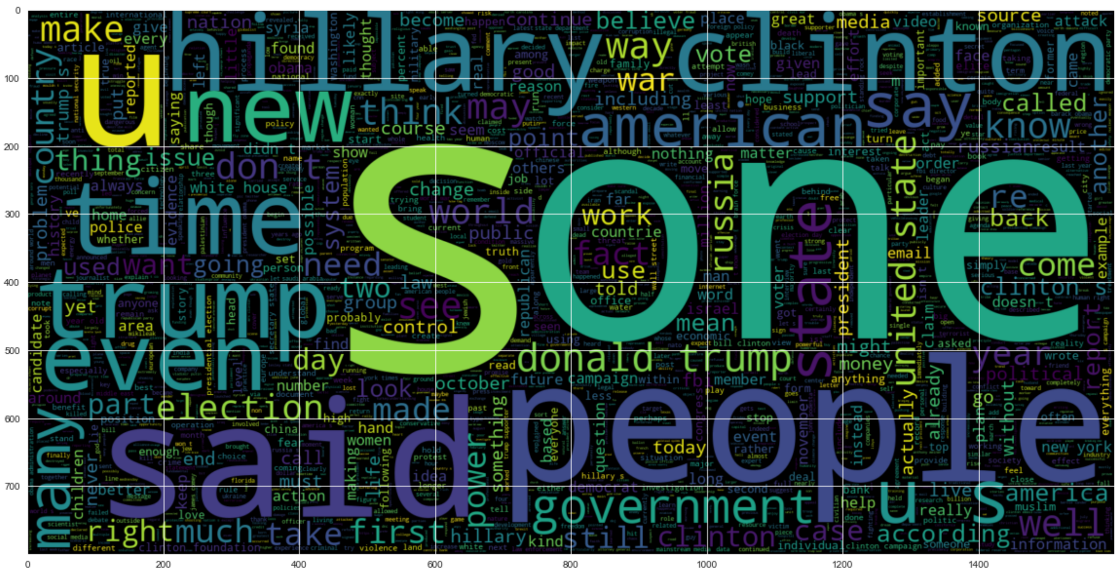


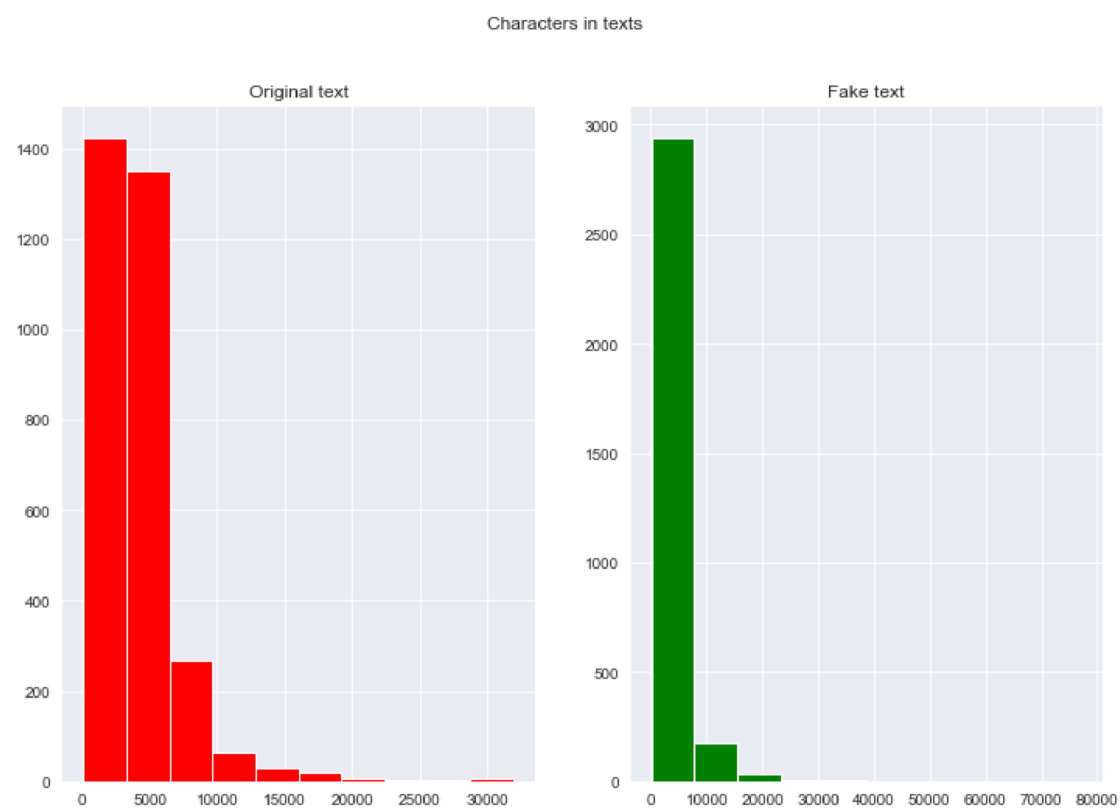


people
one
said
state
american
hillary clinton
say
new
many
trump
even
government
time
donald trump
united state
election
report
thing
power
work
back
country
believe
world
issue
first clinton
clinton s
fact
look
war
never
campaign
year
know
see
take
vote
make
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200
1400

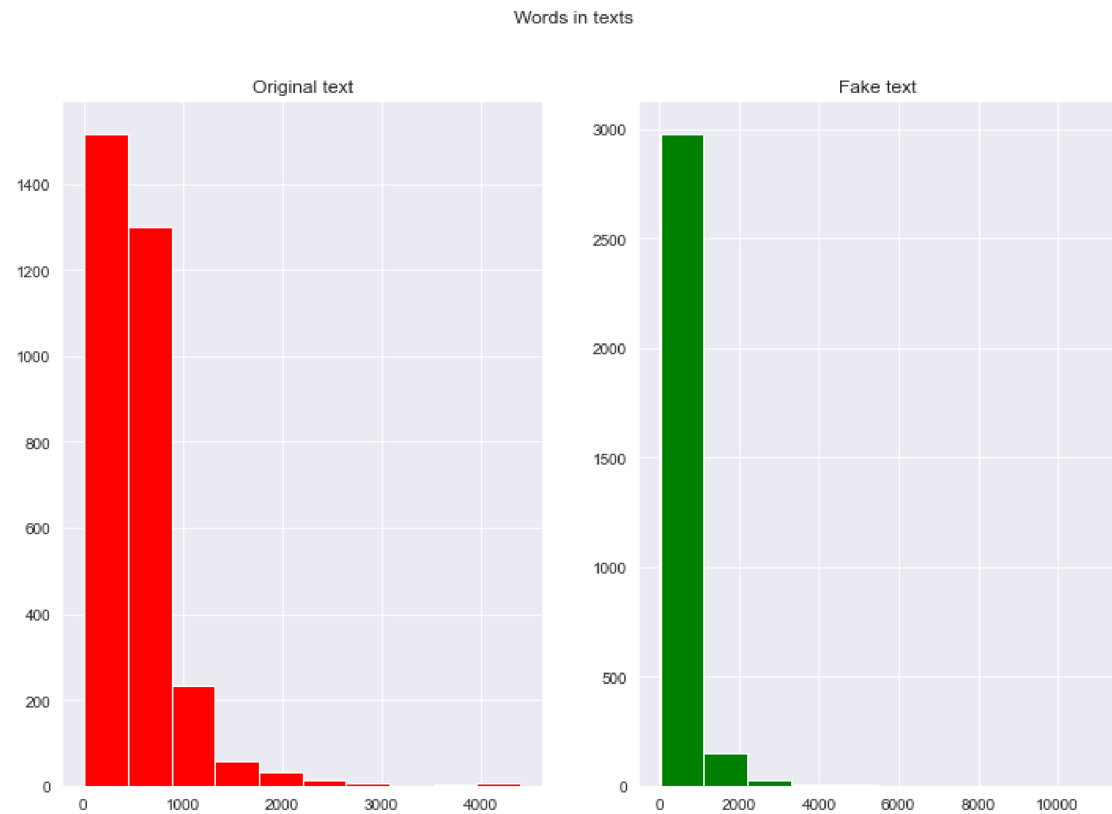


trump
said
clinton
one
say
campaign
people
republican
candidate
donald trump
state
president
think
vote
democrat
even
many
time
obama
hillary clinton
white house
united states
country
american
government
political
debate
new york
support
way
issue
may
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200
1400

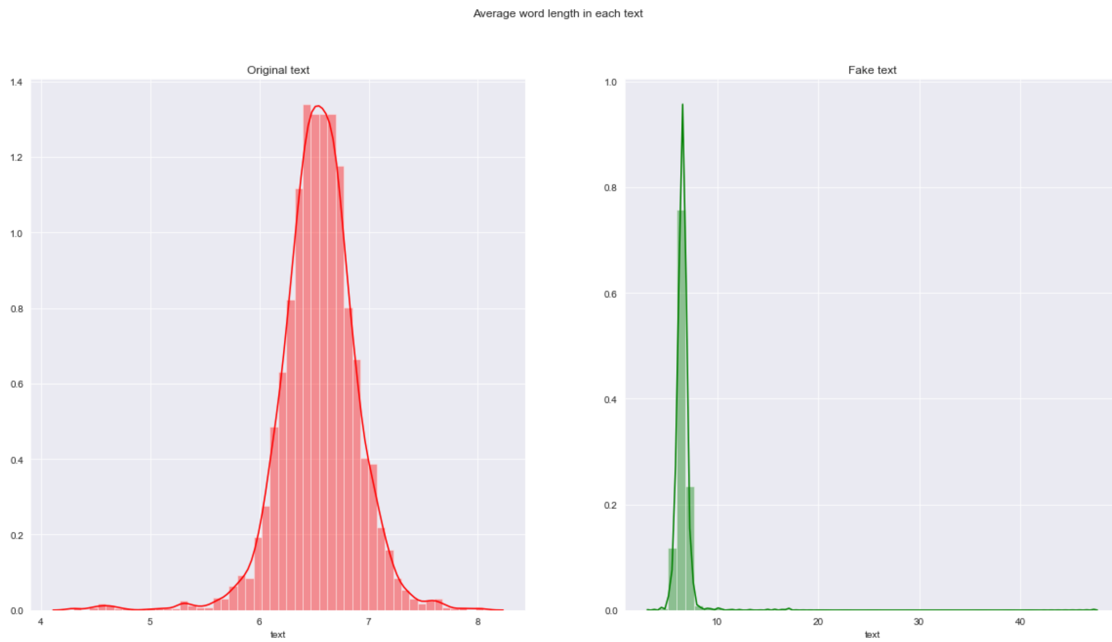


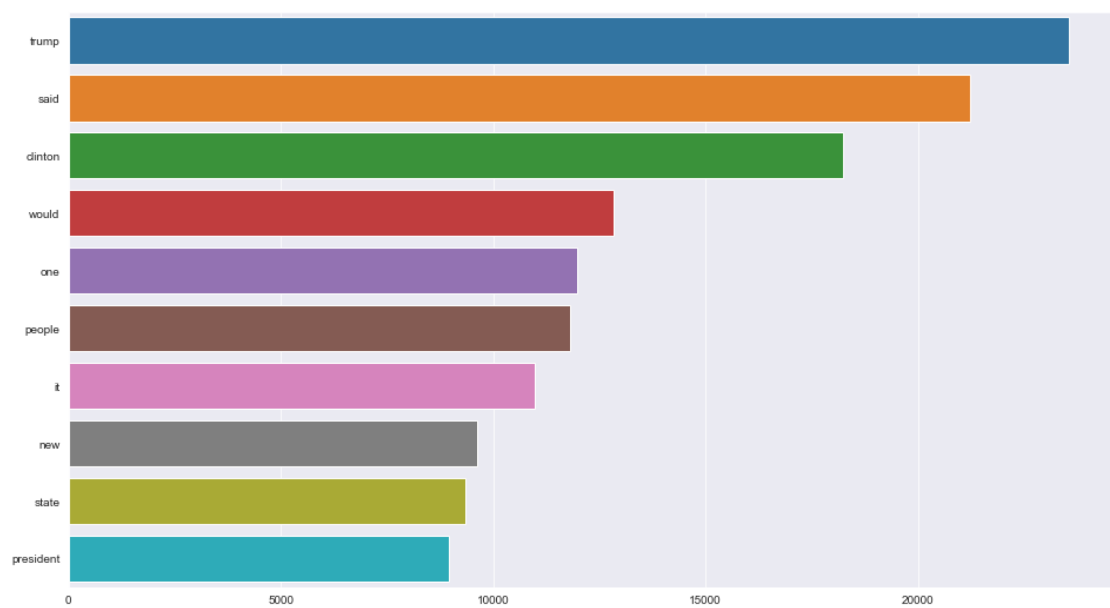


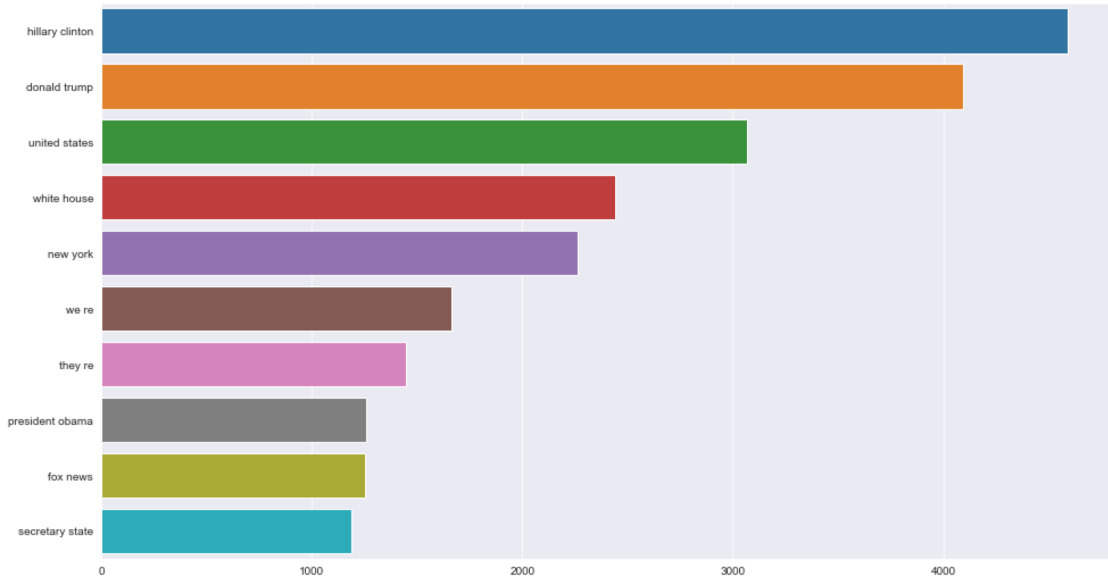
The distribution of both seems to be a bit different.
2500 characters in text is the most common in original text category
while around 5000 characters in text are most common in fake text category.

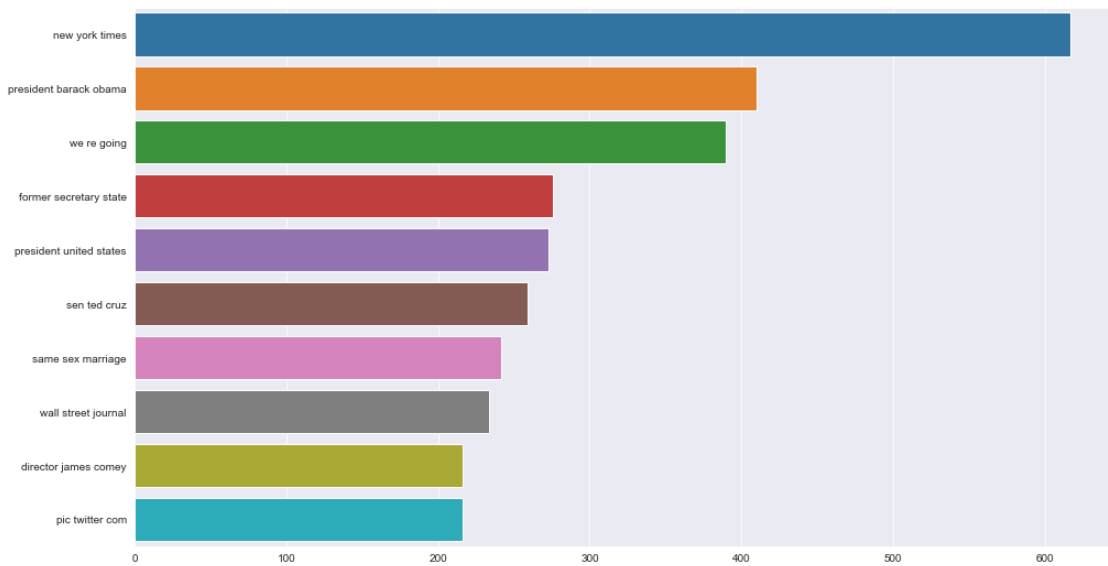



```
Top 5 Words
 ['daniel', 'greenfield,', 'shillman', 'journalism', 'fellow']
Numbers of most common words
 {'trump': 15799, 'said': 13413, 'would': 12593, 'clinton': 12551, 'one': 10293,
'people': 9336, 'new': 9198, '-': 8804, 'also': 8033, 'like': 6646}
```


Average word length in each text
Original text
Fake text
text
text
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
4
5
6
7
8
10
20
30
40


trump
said
clinton
would
one
people
it
new
state
president
0
5000
10000
15000
20000





In order to adjust the model parameters conveniently, we save the cleaned data in the file test2.csv.

[30]:
```
df.to_csv("text2.csv",index=False)
```

[31]:
```
x_test2 = df['text']
y_test2 = df['label']
```

[33]:
```
tfidf_vectorizer = TfidfVectorizer(ngram_range=(1,2), max_features=15000,␣
↪use_idf=True, stop_words='english')
```

```python
x_test2 = tfidf_vectorizer.fit_transform(x_test2)
```

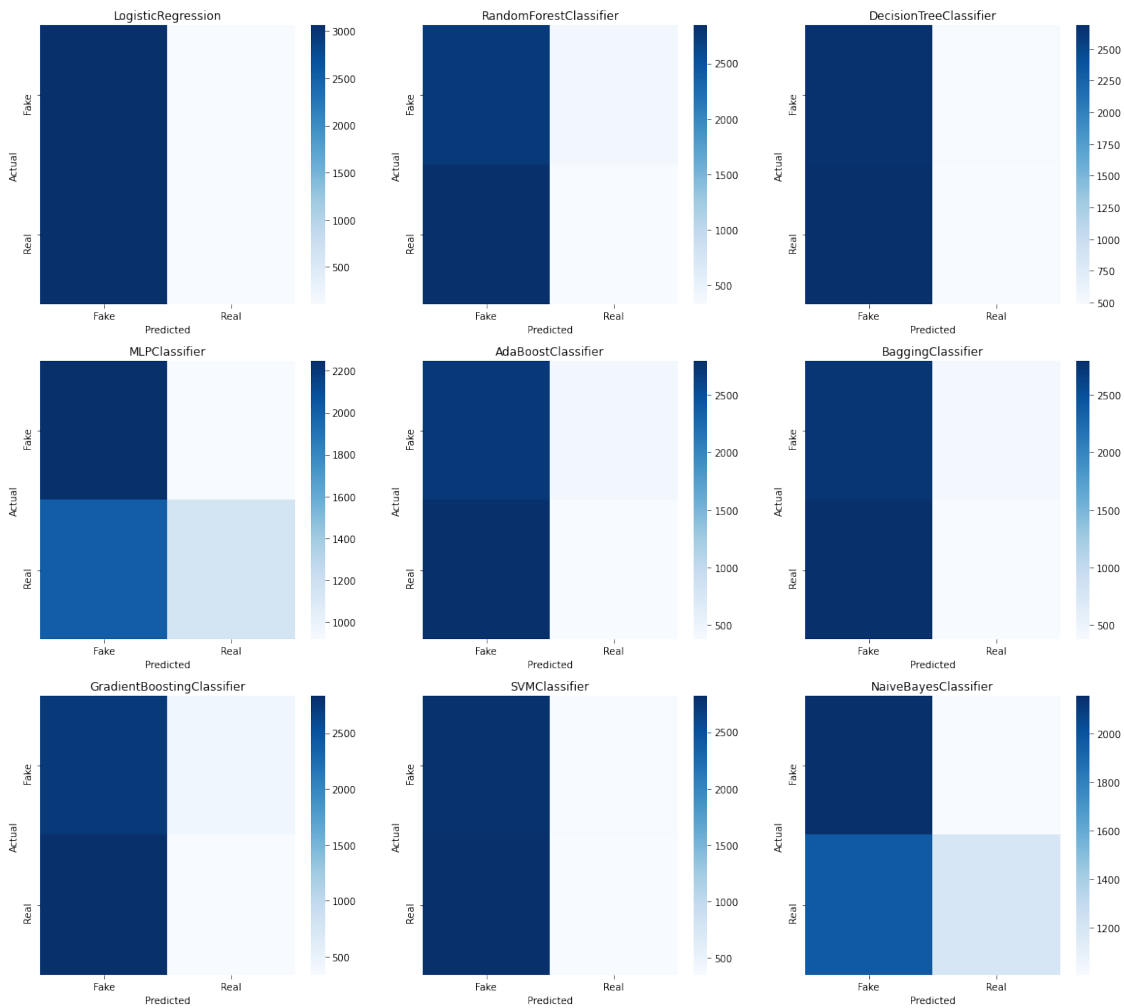
[34]:

```python
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_test2, y_test2))
    y_pred = model.predict(x_test2)
    cm = confusion_matrix(y_test2,y_pred)
    cms.append(cm)
    print(model,'\n',cm)

from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(x_train.toarray(),y_train)
acc.append(model.score(x_test2.toarray(), y_test2))
y_pred = model.predict(x_test2.toarray())
cm = confusion_matrix(y_test2,y_pred)
cms.append(cm)
print('GaussianNB()\n',cm)
```

```
LogisticRegression()
 [[3062  102]
 [3060  111]]
RandomForestClassifier()
 [[2753  411]
 [2843  328]]
DecisionTreeClassifier()
 [[2664  500]
 [2686  485]]
MLPClassifier(max_iter=100)
 [[2250  914]
 [2016 1155]]
AdaBoostClassifier()
 [[2713  451]
 [2798  373]]
BaggingClassifier()
 [[2743  421]
 [2801  370]]
GradientBoostingClassifier()
 [[2733  431]
 [2835  336]]
SVC(kernel='linear')
 [[2801  363]
 [2823  348]]
GaussianNB()
```

```
[[2158 1006]
 [1973 1198]]
```

[35]:
```
fig,ax=plt.subplots(3,3,figsize=(20,18))
for i in range(len(cms)):
    plt.subplot(3, 3, 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])
```



[36]:
```
a = pd.DataFrame({"name": model_name, "acc": acc})
a
```

[36]:

	name	acc
0	LogisticRegression	0.500868
1	RandomForestClassifier	0.486346
2	DecisionTreeClassifier	0.497080
3	MLPClassifier	0.537490
4	AdaBoostClassifier	0.487135
5	BaggingClassifier	0.491397
6	GradientBoostingClassifier	0.484451
7	SVMClassifier	0.497080
8	NaiveBayesClassifier	0.529755

When we test the model on other dataset, **MLP** and **Native Bayes** also have the best performance. So, we only use these two models in future work.

This document is written by team member Yuechen Jiang