Social Media Classification Project

1. Import packages

```
import os
import numpy as np
import pandas as pd
import re
import nltk
nltk.download('punkt')
nltk.download('stopwords')
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
import matplotlib.pyplot as plt
import seaborn as sns
import string
import warnings
warnings.filterwarnings("ignore")

%matplotlib inline
```

First, import packages that I am going to use in this project.

2. Load data

```
df_complaint = pd.read_csv("/Users/kira/Desktop/CIS434 Social Media/Final Project/data/complaint1700.csv")
df_noncomplaint = pd.read_csv("/Users/kira/Desktop/CIS434 Social Media/Final Project/data/noncomplaint1700.csv")
```

```
df_complaint['sentiment'] = 1
df_noncomplaint['sentiment'] = 0
```

```
df = pd.concat([df_complaint,df_noncomplaint],axis=0)
```

```
punctuation = [char for char in string.punctuation if char!='?']
def remove_punc_stopwords(text):
    word_list = nltk.word_tokenize(text)
    not_in_punc = [word for word in word_list if word.lower() not in punctuation]
    return [word.lower() for word in not_in_punc if word.lower() not in stopwords.words(['english','french','spanish','portuguese'])]
```

Load the data that professor uploaded on blackboard and tag the sentiment of each tweet. 1 as negative and 0 as positive.

3. Create Term-Document Matrix(TDM) ¶

```
from sklearn.feature_extraction.text import CountVectorizer
tdm_transformer = CountVectorizer(analyzer=remove_punc_stopwords).fit(df['tweet'])
df_tdm = tdm_transformer.transform(df['tweet'])
```

```
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer().fit(df_tdm)
df_tfidf = tfidf_transformer.transform(df_tdm)
```

Create the TDM.

4. Model Training and Selection

4.1 Split the data

```
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report
from sklearn.model_selection import KFold, cross_val_score, GridSearchCV
from sklearn.metrics import accuracy_score, roc_curve, auc, roc_auc_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from scipy.sparse import hstack
```

```
# create a pipeline to convert the data into tfidf form
pipeline = Pipeline([
    ('bow', CountVectorizer(analyzer=remove_punc_stopwords)),  # strings to token integer counts
    ('tfidf', TfidfTransformer())])
```

```
# Specify X and y
X = pipeline.fit_transform(df.tweet)
y = df.sentiment

# Train test split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.1,random_state=1)
```

In order to train the model, firstly I split the data into training set and validation set. Training set is used to build model and validation set is used to tune the hyperparameter to evaluate the best model.

4.2 evaluation functions

```
def gridSearchCV(model, params):
    """
    @param    model: sklearn estimator
    @param    params (dict): Dictionary of possible parameters

    @return   cv_results (DataFrame)
    """
    model_cv = GridSearchCV(model, param_grid=params, scoring='roc_auc', cv=5)
    model_cv.fit(X_train, y_train)
    cv_results = pd.DataFrame(model_cv.cv_results_)[['params', 'mean_test_score']].sort_values(['mean_test_score'],asc
ending=False)

    return cv_results
```

```
def evaluate(model):
    """
    1. Plot ROC AUC of the test set
    2. Return the best threshold
    """
    model.fit(X_train, y_train)
    probs = model.predict_proba(X_test)
    preds = probs[:,1]
    fpr, tpr, threshold = roc_curve(y_test, preds)
    roc_auc = auc(fpr, tpr)
    print(f'AUC: {roc_auc:.4f}')

    # Find optimal threshold
    rocDf = pd.DataFrame({'fpr': fpr, 'tpr':tpr, 'threshold':threshold})
    rocDf['tpr - fpr'] = rocDf.tpr - rocDf.fpr
    optimalThreshold = rocDf.threshold[rocDf['tpr - fpr'].idxmax()]
    print(optimalThreshold)

    # Get accuracy over the test set
    y_pred = np.where(preds >= optimalThreshold, 1, 0)
    accuracy = accuracy_score(y_test, y_pred)
    print(f'Accuracy: {accuracy*100:.2f}%')
```

gridSearchCV function is used to interate through different hyperparameters to tune the model.
Evaluate function is used to evaluate the performance of each model.

4.3.1 Naive Bayesion

```
nab = MultinomialNB()
params1 = {'alpha': np.linspace(0.5, 1.5, 6), 'fit_prior': [True, False]}
```

```
print(gridSearchCV(nab, params1))
```

```
                                 params  mean_test_score
10   {'alpha': 1.5, 'fit_prior': True}         0.832142
11  {'alpha': 1.5, 'fit_prior': False}         0.832142
8    {'alpha': 1.3, 'fit_prior': True}         0.831523
9   {'alpha': 1.3, 'fit_prior': False}         0.831523
6    {'alpha': 1.1, 'fit_prior': True}         0.830769
7   {'alpha': 1.1, 'fit_prior': False}         0.830769
4    {'alpha': 0.9, 'fit_prior': True}         0.829392
5   {'alpha': 0.9, 'fit_prior': False}         0.829392
2    {'alpha': 0.7, 'fit_prior': True}         0.827539
3   {'alpha': 0.7, 'fit_prior': False}         0.827539
0    {'alpha': 0.5, 'fit_prior': True}         0.823701
1   {'alpha': 0.5, 'fit_prior': False}         0.823701
```

```
nab = MultinomialNB(alpha=1.5,fit_prior=True)
evaluate(nab)
```

```
AUC: 0.8236
0.5590393242672023
Accuracy: 77.35%
```

The first model I trained is Naive Bayesion. By tuning 'alpha' and 'fit_prior', I found the best hyperparameters for this model, and evaluated this model's performance.

4.3.2 Random Forest

```
params1 = {'bootstrap': [True, False]}
params2 = {'max_depth': [None,1,2,5,8,10,20,30,40,50,60,70,80,90,100]}
params3 = {'n_estimators': [2,4,6,8,10,12,14,16,18,20,30,40,50,60,70,80,90,100]}
params4 = {'max_features': [None,1,2,5,8,10,20,30,40,50,60,70,80,90,100]}
```

```
rfc = RandomForestClassifier(random_state=1)
print(gridSearchCV(rfc, params1))
```

```
                  params  mean_test_score
1  {'bootstrap': False}         0.767762
0   {'bootstrap': True}         0.766037
```

```
rfc = RandomForestClassifier(random_state=1,bootstrap=False)
print(gridSearchCV(rfc, params2))
```

```
                  params  mean_test_score
13     {'max_depth': 90}         0.780487
11     {'max_depth': 70}         0.777657
14    {'max_depth': 100}         0.777026
10     {'max_depth': 60}         0.774856
9      {'max_depth': 50}         0.773707
12     {'max_depth': 80}         0.773058
0    {'max_depth': None}         0.767762
7      {'max_depth': 30}         0.765512
8      {'max_depth': 40}         0.764232
6      {'max_depth': 20}         0.745030
5      {'max_depth': 10}         0.706911
4       {'max_depth': 8}         0.689439
3       {'max_depth': 5}         0.647626
2       {'max_depth': 2}         0.575075
1       {'max_depth': 1}         0.527678
```

```
rfc = RandomForestClassifier(random_state=1,bootstrap=False,max_depth=90)
print(gridSearchCV(rfc, params3))
```

```
                     params  mean_test_score
16   {'n_estimators': 90}         0.812094
17  {'n_estimators': 100}         0.812069
15   {'n_estimators': 80}         0.810814
14   {'n_estimators': 70}         0.810357
13   {'n_estimators': 60}         0.809247
12   {'n_estimators': 50}         0.806923
11   {'n_estimators': 40}         0.805249
10   {'n_estimators': 30}         0.801320
9    {'n_estimators': 20}         0.795087
8    {'n_estimators': 18}         0.793189
7    {'n_estimators': 16}         0.792039
6    {'n_estimators': 14}         0.789643
5    {'n_estimators': 12}         0.784776
4    {'n_estimators': 10}         0.780487
3     {'n_estimators': 8}         0.774214
2     {'n_estimators': 6}         0.760836
1     {'n_estimators': 4}         0.744637
0     {'n_estimators': 2}         0.696660
```

```
rfc = RandomForestClassifier(random_state=1,bootstrap=False,max_depth=90,n_estimators=90)
print(gridSearchCV(rfc, params4))
```

```
                      params  mean_test_score
8     {'max_features': 40}         0.818822
10    {'max_features': 60}         0.814319
9     {'max_features': 50}         0.813710
7     {'max_features': 30}         0.812109
14   {'max_features': 100}         0.811849
11    {'max_features': 70}         0.811779
12    {'max_features': 80}         0.811688
6     {'max_features': 20}         0.811094
13    {'max_features': 90}         0.810468
5     {'max_features': 10}         0.800087
4      {'max_features': 8}         0.798953
3      {'max_features': 5}         0.777705
2      {'max_features': 2}         0.749813
1      {'max_features': 1}         0.724272
0   {'max_features': None}         0.685222
```

```
rfc = RandomForestClassifier(random_state=1,bootstrap=True,max_depth=90,n_estimators=90,max_features=40)
evaluate(rfc)
```

```
AUC: 0.8057
0.488788321880853
Accuracy: 75.88%
```

Next, using the same process, I found the best Random forest model.

4.3.3 SVM

```
params1 = {'C': [0.001,0.01,0.1,1,3,10,20,30,40,50],
           'kernel':['linear','rbf','poly']}
params2 = {'gamma':[0.001,0.01,0.1,1,10,100]}
params3 = {'degree':[0,1,2,3,4,5,6,7,8,9,10,20,30,40,50]}
```

```
svc = SVC()
print(gridSearchCV(svc, params1))
```

```
                              params  mean_test_score
9       {'C': 1, 'kernel': 'linear'}         0.820981
5      {'C': 0.01, 'kernel': 'poly'}         0.813005
23       {'C': 30, 'kernel': 'poly'}         0.812924
17       {'C': 10, 'kernel': 'poly'}         0.812867
29       {'C': 50, 'kernel': 'poly'}         0.812848
26       {'C': 40, 'kernel': 'poly'}         0.812809
8       {'C': 0.1, 'kernel': 'poly'}         0.812795
20       {'C': 20, 'kernel': 'poly'}         0.812782
14        {'C': 3, 'kernel': 'poly'}         0.812768
11        {'C': 1, 'kernel': 'poly'}         0.812750
6     {'C': 0.1, 'kernel': 'linear'}         0.802755
12      {'C': 3, 'kernel': 'linear'}         0.801027
0   {'C': 0.001, 'kernel': 'linear'}         0.798804
13         {'C': 3, 'kernel': 'rbf'}         0.798804
10         {'C': 1, 'kernel': 'rbf'}         0.798804
16        {'C': 10, 'kernel': 'rbf'}         0.798776
25        {'C': 40, 'kernel': 'rbf'}         0.798774
28        {'C': 50, 'kernel': 'rbf'}         0.798774
22        {'C': 30, 'kernel': 'rbf'}         0.798774
3    {'C': 0.01, 'kernel': 'linear'}         0.798772
19        {'C': 20, 'kernel': 'rbf'}         0.798759
4       {'C': 0.01, 'kernel': 'rbf'}         0.798561
1      {'C': 0.001, 'kernel': 'rbf'}         0.798554
7        {'C': 0.1, 'kernel': 'rbf'}         0.798405
15     {'C': 10, 'kernel': 'linear'}         0.786872
18     {'C': 20, 'kernel': 'linear'}         0.785597
24     {'C': 40, 'kernel': 'linear'}         0.785571
27     {'C': 50, 'kernel': 'linear'}         0.785571
21     {'C': 30, 'kernel': 'linear'}         0.785571
2     {'C': 0.001, 'kernel': 'poly'}         0.775101
```

```
svc = SVC(C=1,kernel='linear')
print(gridSearchCV(svc, params2))
```

```
              params  mean_test_score
0  {'gamma': 0.001}         0.820981
1   {'gamma': 0.01}         0.820981
2    {'gamma': 0.1}         0.820981
3      {'gamma': 1}         0.820981
4     {'gamma': 10}         0.820981
5    {'gamma': 100}         0.820981
```

```
svc = SVC(C=1,kernel='linear',gamma=0.001)
print(gridSearchCV(svc, params3))
```

```
            params  mean_test_score
0    {'degree': 0}         0.820981
1    {'degree': 1}         0.820981
2    {'degree': 2}         0.820981
3    {'degree': 3}         0.820981
4    {'degree': 4}         0.820981
5    {'degree': 5}         0.820981
6    {'degree': 6}         0.820981
7    {'degree': 7}         0.820981
8    {'degree': 8}         0.820981
9    {'degree': 9}         0.820981
10  {'degree': 10}         0.820981
11  {'degree': 20}         0.820981
12  {'degree': 30}         0.820981
13  {'degree': 40}         0.820981
14  {'degree': 50}         0.820981
```

```
svc = SVC(random_state=1,C=1,kernel='linear',gamma=0.001,degree=0,probability=True)
evaluate(svc)
```

```
AUC: 0.8258
0.5256810441625582
Accuracy: 78.24%
```

The best SVM model.

In the end, we can see that linear SVM model with C=1, gamma=0.001, degree=0 shows the best accuracy at 78.24%.
I chose this as the Final model.

5. Final Model

```
df_test = pd.read_csv("/Users/kira/Desktop/CIS434 Social Media/Final Project/data/tweet_test.csv")
df_test = df_test.drop(['tid_not_to_be_used','airline','tag'],axis=1)
```

```
final_model = SVC(random_state=1,C=1,kernel='linear',gamma=0.001,degree=0,probability=True)
final_model.fit(X,y)
```

```
SVC(C=1, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=0, gamma=0.001, kernel='linear',
    max_iter=-1, probability=True, random_state=1, shrinking=True, tol=0.001,
    verbose=False)
```

```
final_X = pipeline.transform(df_test.tweet)
predictions = final_model.predict_proba(final_X)[:,1]
y_pred = np.where(predictions >= 0.15, 1, 0)
```

```
output = pd.DataFrame({
    'id':df_test.id,
    'tweet':df_test.tweet,
    'pred':y_pred
})
```

```
res = output[output['pred']==0]
res = res[['id','pred','tweet']]
res = res.reset_index(drop=True)
res
```

	id	pred	tweet
0	564	0	Shoutout to Crystal at @JetBlue for helping us...
1	620	0	.@richardbranson .@rmchrQB .@VirginAmerica Air...
2	2291	0	On @jetblue heading to Vegas for my first @ABC...
3	2338	0	Can't wait to fly @JetBlue #TrueBlue
4	2523	0	When did @AlaskaAir become the most expensive ...
...	...	...	...
288	170610	0	Best way to leave @FlyTPA @JetBlue be right ba...
289	170650	0	So sad @united https://t.co/2JP5WXlpd7
290	172276	0	Hey you guys, @JetBlue is the best. Seriously....
291	172719	0	@Charalanahzard that's why you never use @Amer...
292	172913	0	@AbdulNasirJ @HussainKamani @united Prophet fu...

293 rows × 3 columns

This the dataframe containing the prediction generated by my final model. The outcome is stored in 'pred' column.

```
res = res.drop(['pred'],axis=1)
```

```
res.to_csv(r'/Users/kira/Desktop/CIS434 Social Media/Final Project/Geng_Luo.csv',sep=',')
```

```
my_eval = pd.read_csv("/Users/kira/Desktop/CIS434 Social Media/Final Project/my_eval.csv")
```

```
my_eval
```

	my_eval
0	1
1	0
2	1
3	1
4	0
...	...
288	1

Then, I kept only positive tweets and dropped the column 'pred' and saved the dataframe as csv file to self-eveluate the result. Self-evaluated result is saved into my_eval.csv file, so I read csv as dataframe again and added column my_eval into the dataframe.

```
res['my_eval'] = my_eval
res = res[['id','my_eval','tweet']]
res
```

	id	my_eval	tweet
0	564	1	Shoutout to Crystal at @JetBlue for helping us...
1	620	0	.@richardbranson .@rmchrQB .@VirginAmerica Air...
2	2291	1	On @jetblue heading to Vegas for my first @ABC...
3	2338	1	Can't wait to fly @JetBlue #TrueBlue
4	2523	0	When did @AlaskaAir become the most expensive ...
...	...	...	...
288	170610	1	Best way to leave @FlyTPA @JetBlue be right ba...
289	170650	1	So sad @united https://t.co/2JP5WXlpd7
290	172276	1	Hey you guys, @JetBlue is the best. Seriously....
291	172719	1	@Charalanahzard that's why you never use @Amer...
292	172913	0	@AbdulNasirJ @HussainKamani @united Prophet fu...

293 rows × 3 columns

```
res.to_csv(r'/Users/kira/Desktop/CIS434 Social Media/Final Project/Geng_Luo.csv',sep=',')
```

Final accuracy was 74.4%.