CIS 434 Final Project by Wenjie Li

To predict whether a tweet is a complaint, I decide to analyze whether the test tweet contains specific terms that contribute to a complaint tweet. For this purpose, I use TFIDF. Below is the breakdown of my method.First, I import nececcary packages.

In [1]:

```
import string
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
ps = PorterStemmer()
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
import csv
import numpy as np
from string import digits
from sklearn.linear_model import LogisticRegression
from sklearn import svm
```

Then I read training data, both complaints and non-complaints, and drop useless columns.

In [2]:

```
# read training data
complaints = pd.read_csv('complaint1700.csv')
non_complaints = pd.read_csv('noncomplaint1700.csv')
# extract tweets
complaints = complaints.drop(['id', 'airline'], axis = 1)
non_complaints = non_complaints.drop(['id', 'airline'], axis = 1
)
```

Since the original data does not contain label values, I need to add manually 'complaint' and 'non-complaint', which are binary values.

In [3]:

```
# add label
complaints['label'] = 'complaint'
non_complaints['label'] = 'non-complaint'
```

Now let me combine the complaint and non-complaint data. This is the dataset to use before preprocessing.

In [4]:

```
# combine
data = complaints.append(non_complaints)
# reset index
data.index = range(3400)
```

Tweets in this dataset include redundant information, including punctuation, stopwords, which serve no values but grammatical ones, and verb-ing, verb-ed, etc. These must be eliminated before any analytical work.

In [5]:

```
# standardize tweet: remove punctuation, stopwords & stem
def text_process(mess):
    """
    Takes in a string of text, then performs the following:
    1. Remove all punctuation
    2. Remove all stopwords
    3. Returns a list of the cleaned text
    """
    # Check characters to see if they are in punctuation
    nopunc = [char for char in mess if char not in string.punctu
ation]

    # Join the characters again to form the string.
    nopunc = ''.join(nopunc)

    # Now just remove any stopwords
    nostop_tweet = [word for word in nopunc.split() if word.lowe
r() not in stopwords.words('english')]

    # stem
    stem_tweet = [ps.stem(word) for word in nostop_tweet]

    final_tweet = ' '.join(str(i) for i in stem_tweet)

    remove_digits = str.maketrans('', '', digits)
    res = final_tweet.translate(remove_digits)
    return(res)
```

In [6]:

```
# apply text_process to tweets
data['tweet'] = data['tweet'].apply(text_process)
```

If tweets remain in their original form, which is natural language, there is no way that computer can do processiong work. One way to handle this is to vectorize each tweet into categorical values. One precondition of vectorization is to create bag of words, meaning all the words appearing in tweets in the training data. After vectorization, I add weight to every word in each tweet to reflect the importance of that word. In this way TFIDF is created, after which I can start out training models. I choose MultinomialNB and

incorporate it into Pipeline along with vectorization and TFIDF functions.

In [7]:

```
pipeline = Pipeline([
    ('bow', CountVectorizer(analyzer=text_process)),  # strings to token integer counts
    ('tfidf', TfidfTransformer()),  # integer counts to weighted TF-IDF scores
    ('classifier', LogisticRegression()),  # train on TF-IDF vectors w/ Naive Bayes classifier
])

pipeline.fit(data['tweet'],data['label'])
```

```
/Users/johnlee/anaconda3/lib/python3.7/site-packages
/sklearn/linear_model/logistic.py:432: FutureWarning
: Default solver will be changed to 'lbfgs' in 0.22.
Specify a solver to silence this warning.
  FutureWarning)
```

Out[7]:

```
Pipeline(memory=None,
         steps=[('bow',
                 CountVectorizer(analyzer=<function
text_process at 0x1a21a02290>,
                                 binary=False, decod
e_error='strict',
                                 dtype=<class 'numpy
.int64'>, encoding='utf-8',
                                 input='content', lo
wercase=True, max_df=1.0,
                                 max_features=None,
min_df=1,
                                 ngram_range=(1, 1),
preprocessor=None,
                                 stop_words=None, st
rip_accents=None,
                                 token_pattern='(?u)
\\b\\w\\w+\\b',...
                ('tfidf',
                 TfidfTransformer(norm='l2', smooth_
idf=True,
                                  sublinear_tf=False
, use_idf=True)),
                ('classifier',
                 LogisticRegression(C=1.0, class_wei
ght=None, dual=False,
                                    fit_intercept=Tr
ue, intercept_scaling=1,
                                    l1_ratio=None, m
ax_iter=100,
                                    multi_class='war
n', n_jobs=None,
                                    penalty='l2', ra
ndom_state=None,
                                    solver='warn', t
ol=0.0001, verbose=0,
                                    warm_start=False
))],
         verbose=False)
```

Since the model is now trained, I should test the model on the test data.

In [8]:

```
# read test data
test_data = pd.read_csv('test_data.csv')
```

In [9]:

```
# predict and assign complaint and non-complaint to each tweet.
I use 0.7 mainly considering the efficiency of manual checking i
n the following
# part.
predictions = pipeline.predict_proba(test_data['tweet'])
y_pred = np.where(predictions[:, 1] >= 0.7, 'non-complaint', 'co
mplaint')
```

Now it is time to build the csv file required by the professor.

In [10]:

```
dict = {}
dict['id'] = test_data['id'][y_pred == 'non-complaint']
dict['evaluation'] = np.zeros(154)
dict['non_complaint_tweets'] = test_data['tweet'][y_pred == 'non
-complaint']
df = pd.DataFrame(data = dict, columns = ['id', 'evaluation', 'n
on_complaint_tweets'])
df.to_csv('pred.csv',sep=',',index=False,header=False)
```

Next, I manually check whether each tweet that is predicted to be non-complaint is actually non-complaint. For convenience, in the csv file, I refer complaint to be 0 and non-complaint to be 1. It turns out that out of the 154 tweets, 78 are precise, bringing about a precision of 51%.