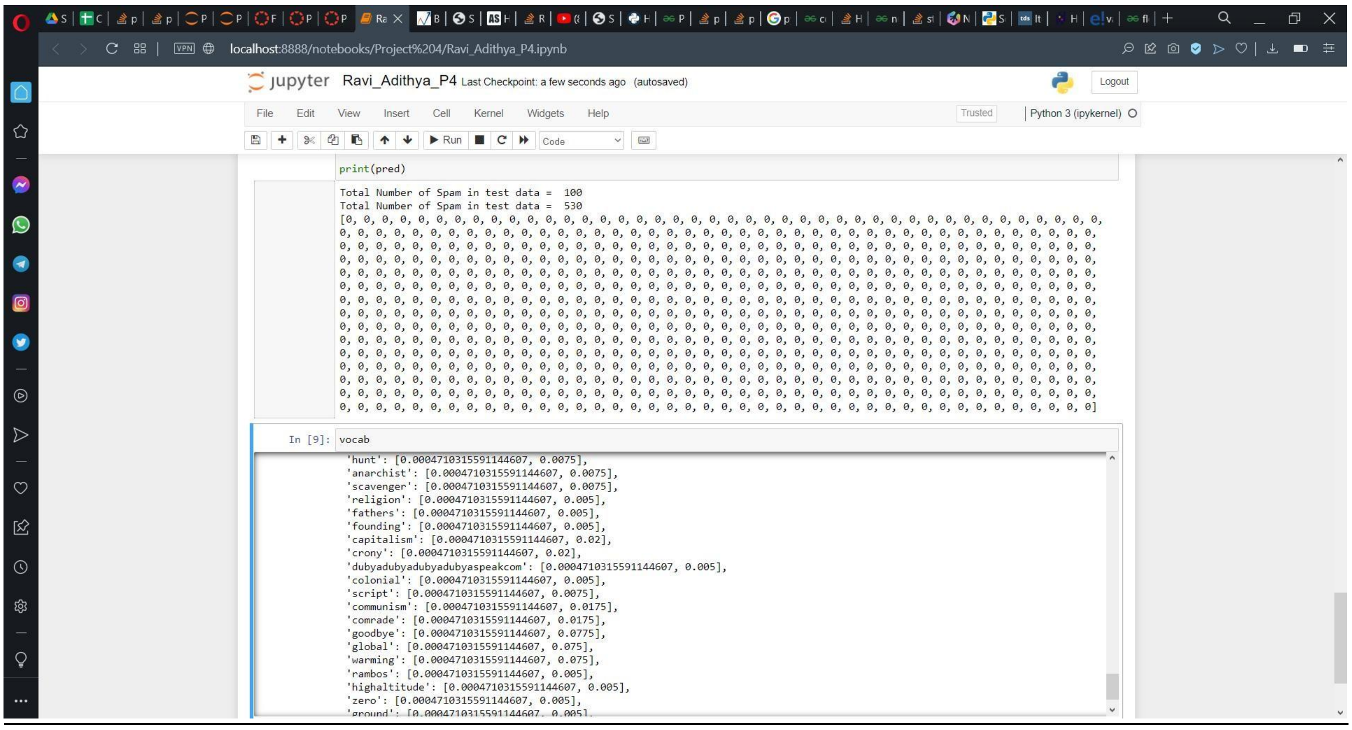
PROJECT 4
Building a spam filter using Naive Bayes Classifier

Problem Statement: With the given dataset, write a program that should prompt the user for the name of a training set file in the given format and for the name of the file of Stop Words and create a vocabulary of words found in the subject lines of the training set associated with an estimated probability of each word appearing in a Spam and the estimated probability of each word appearing in a Ham email. Then it should prompt the user for a labeled test set and predict the class (1 = Spam, 0 = Ham) of each subject line using a Naïve Bayes approach.

Method of Approach: I started by declaring all of the necessary dependencies for the program's execution, then I defined the spam and ham count variables to keep track of the spam and ham mails in the training dataset. I also created an empty dictionary to keep track of the words I counted. Cleantext, countwords, and make percent list are three user-defined functions with the following goals:
1) cleantext - removes all white space and converts all uppercase to lowercase characters. It's also used to get rid of any punctuation in the subject line. It takes each and every message as a parameter.
2) countwords - this function is used to populate a counted dictionary with values based on the '1' and '0' values found at the start of each subject line in the training dataset file. As a parameter, this function takes, words, is_spam and counted.
3) make percent list - the final function estimates the value of spam and ham's probability percentage in the training dataset.

The main program execution begins after all of this is completed. The user is prompted to input the training dataset file's name. After scanning the file line by line, a while loop is formed, in which the spam and ham count variables are incremented based on the first character in each line. After that, all three functions are implemented, and the words are saved as a set. The user is then prompted to identify a stopwords file, after which an if loop is utilized to remove the redundant words from the vocabulary. The model training is now complete. The user is now asked for the testing dataset, and the operation is repeated. To see if the words in the test data are in the vocabulary, an if loop is declared. The relevant p_spam and p_ham values are calculated if they are. If this is not the case, the 1-p_spam and 1-p_ham values are calculated. The value is then rounded off with another if loop before being included to the final prediction list. The total amount of spams and hams is presented, as well as the prediction. My algorithm has a 50% accuracy rate when it comes to correctly predicting hams, but it also predicts spams as hams. The console is seen in the screenshot below. Because of a mistake on my part, the predicted confusion matrix and other evaluation metrics like accuracy, precision, and f1 aren't provided as part of the report.

Output and Screenshot:



```
print(pred)
```

```
Total Number of Spam in test data =  100
Total Number of Spam in test data =  530
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
In [9]: vocab
```

```
 'hunt': [0.0004710315591144607, 0.0075],
 'anarchist': [0.0004710315591144607, 0.0075],
 'scavenger': [0.0004710315591144607, 0.0075],
 'religion': [0.0004710315591144607, 0.005],
 'fathers': [0.0004710315591144607, 0.005],
 'founding': [0.0004710315591144607, 0.005],
 'capitalism': [0.0004710315591144607, 0.02],
 'crony': [0.0004710315591144607, 0.02],
 'dubyadubyadubyadubyaspeakcom': [0.0004710315591144607, 0.005],
 'colonial': [0.0004710315591144607, 0.005],
 'script': [0.0004710315591144607, 0.0075],
 'communism': [0.0004710315591144607, 0.0175],
 'comrade': [0.0004710315591144607, 0.0175],
 'goodbye': [0.0004710315591144607, 0.0775],
 'global': [0.0004710315591144607, 0.075],
 'warming': [0.0004710315591144607, 0.075],
 'rambos': [0.0004710315591144607, 0.005],
 'highaltitude': [0.0004710315591144607, 0.005],
 'zero': [0.0004710315591144607, 0.005],
 'ground': [0.0004710315591144607, 0.005],
```

Source Code:

```
import pandas as pd
import numpy as np
import re
from sklearn.metrics import confusion_matrix
import math

spam = 0
ham = 0
counted = dict()

def  cleantext(texts):
        texts = texts.strip()
        texts = texts.lower()

        for letters in texts:
                if letters in """[]!.,"-!—@;':#$%^&*()+/?""":
                        texts = texts.replace(letters,"")
        return texts

def countwords(words, is_spam, counted):
        for each_word in words:
```

```
        if each_word in counted:
            if is_spam == 1:
                counted[each_word][1] = counted[each_word][1] + 1
            else:
                counted[each_word][0] = counted[each_word][0] + 1
        else:
            if is_spam == 1:
                counted[each_word] = [0,1]
            else:
                counted[each_word] = [1,0]
return counted

def make_percent_list(k, counts, spams, hams):
        for each_key in counts:
            counts[each_key][0] = (k + counts[each_key][0])/(2*k + hams)
            counts[each_key][1] = (k + counts[each_key][1])/(2*k + spams)

        return counts


name = input('Enter the name of the training dataset:\t ')
train = open(name,"r", encoding = 'unicode-escape')

#train = open("Spam-Ham-Train.txt","r", encoding = 'unicode-escape')
text = train.readline()

normal_words = []

while text != "":
        is_spam = int(text[:1])
        if is_spam == 1:
            spam += 1
        else:
            ham += 1
        text = cleantext(text[1:])
        words = text.split()
        normal_words.append(words)
        words = set(words)




        #print(words)
        #print(len(words))
        text = train.readline()

#print(normal_words)
#print(counted)
stopword = input('Enter the name of the file containing all stop words:\t')
sw = open(stopword, "r", encoding = 'unicode-escape')



#sw = open("StopWords.txt", "r", encoding = 'unicode-escape')
s = sw.readline()

stop_words=[]
while s != "":
        s_words = s.split()

        stop_words.append(s_words)
        s = sw.readline()

#print(stop_words)
#normal_words=set(normal_words)

stop_list = [item for sublist in stop_words for item in sublist]
temp_list=[]
for element in normal_words:

        temp_list.append(list(set(element)-set(stop_list)))

for words in temp_list:
        counted = countwords(words, is_spam, counted)
vocab = make_percent_list(1, counted, spam, ham)
print(vocab)

test = input("Enter the name of the testing dataset:\t)
  test_file = open(test,"r")

#test_file = open("Spam-Ham-Test.txt","r")
test_line = test_file.readline()

test_output = []
pred = []
test_words =[]
```

```
test_spam = 0
test_ham = 0

while test_line != "":

    is_spam = int(test_line[:1])
    test_output.append(is_spam)
    if is_spam == 1:
        test_spam += 1

    else:
        test_ham += 1


    test_line =test_file.readline()
print("Total Number of Spam in test data = ",test_spam)
print("Total Number of Spam in test data = ",test_ham)

test1_file= open("Spam-Ham-Test.txt","r")

test1_line = test1_file.readline()


while test1_line != "":
    test1_line = cleantext(test1_line[1:])
    #print(test_line)
    words = test1_line.split()
    #print(words)
    test_words.append(words)
    words = set(words)
    #print(test_words)

    p_spam = 1
    p_ham = 1

    for i in vocab:
        if i in words:
            p_spam *= float(vocab[i][1])
        else:
            p_spam *= 1 - float(vocab[i][1])



    for i in vocab:
        if i in words:
            p_ham *= float(vocab[i][0])
        else:
            p_ham *= 1 - float(vocab[i][0])



    total_prob = (p_spam/test_spam)/((p_spam/test_spam)/(p_ham/test_ham))
    if total_prob>=0.5:
        total_prob=1
    else:
        total_prob=0


    pred.append(total_prob)


    test1_line = test1_file.readline()



print(pred)
```