Homework 1

1 Online Shopping Case

First of all, , everything given in this question were written in a table to make the work easier. The table is shown below which is Table-1. For all the points of the question Table-1 was used and other tables were created to see the answers better.

	Product Percentage	Positive Feedback	Negative Feedback
Popular	$P(P) = 45\%$	$P(F_P\|P) = 95\%$	$P(F_N\|P)$ =5%
Moderately Sold	$P(M)$ =30%	$P(F_P\|M) = 60\%$	$P(F_N\|M)$ =40 %
Unpopular	$P(U)$ =25%	$P(F_P\|U) = 10\%$	$P(F_N\|U)$ =90%

Table 1: Given Statistics

1.1

What is the probability that a product gets positive feedback $[P(F_P)]$?

In order to calculate the wanted positive feedback probability $P(F_P)$, product rule and marginalization were used.
By using product rule the equation below is obtained.

$$\sum_{K=Category} P(F_P, K) = \sum_{K=Category} P(F_P|K)P(K)$$

By using marginalization the equation below is obtained.

$$P(F_P) = \sum_{K=Category} P(F_P, K)$$

Using Table-1, $P(F_P|K)$, and $P(K)$ are found ($P(P) = 45\%$, $P(M) = 30\%$, $P(U) = 25\%$, $P(F_P|P) = 95\%$, $P(F_P|M) = 60\%$, $P(F_P|U) = 10\%$) and finally the result is calculated as following:

$$P(F_P) = \frac{45}{100}\frac{95}{100} + \frac{30}{100}\frac{60}{100} + \frac{25}{100}\frac{10}{100} = \frac{6325}{10000} = 63.25\%$$

1.2

If a new product gets positive feedback, what is the probability that it will be a popular product $[P(P|F_P)]$?

In order to calculate the wanted probability $P(P|F_P)$, Product Rule was used.
By using product rule the equation below is obtained.

$$P(P|F_P)P(F_P) = P(P, F_P) = P(F_P|P)P(P)$$

$$P(P|F_P) = \frac{P(F_P|P)P(P)}{P(F_P)}$$

Using Table-1, $P(F_P|P)$, and $P(P)$ were found and $P(F_P)$ (63.25%) was found in part 1.1. Finally the result is calculated as following:

$$P(P|F_P) = \frac{\frac{95}{100}\frac{45}{100}}{\frac{63.25}{100}} = \frac{4275}{6325} \approx 67.5889\%$$

1.3

If a product does not get positive feedback, what is the probability that it will be a popular product $[P(P|F_N)]$?

In order to calculate the wanted probability $P(P|F_N)$, Product Rule was used.
By using product rule the equation below is obtained.

$$P(P|F_N)P(F_N) = P(P, F_N) = P(F_N|P)P(P)$$

$$P(P|F_N) = \frac{P(F_N|P)P(P)}{P(F_N)}$$

Keeping in mind that if a product doesn't get a positive feedback, it takes negative. Therefore $P(F_N)$ and $P(F_P)$ complement each other. That is to say $P(F_N) = 1 - P(F_P) = 1 - 0.6325 = 0.3675$. Also using Table-1, and $P(P)$ was found. Finally the result is calculated as following:

$$P(P|F_N) = \frac{\frac{5}{100}\frac{45}{100}}{\frac{36.75}{100}} = \frac{225}{3675} \approx 6.1224\%$$

2 Spam Email Detection

In this part of the homework we are expected to examine given data files (*x_train.csv*, *y_train.csv*, *x_test.csv*, *y_test.csv*) and answer the asked questions.

2.1 Balanced or Skewed

2.1.1

The total number of spam e-mails in the training set is 1183 and the total number of e-mails are 4137 in the training set. (The number of spam mails are found using the argwhere method of numpy. Even if I could use a loop to find the total number of spam mails, I preferred argwhere method since its runtime was smaller.) Therefore, the spam email ratio is 28.59%.

2.1.2

According to number of spam and ham e-mails, it is seen that training set is skewed towards normal e-mails so we don't have a balanced data set. Having an imbalanced data set would affect the model in a way that the model will be biased towards the normal mails.

2.1.3

As mentioned, having an imbalanced data set towards normal mails, will affect the model in a way that the model will be biased towards the normal mails. This is because the prior probability of having a normal e-mail is higher than having a spam e-mail, so there is a high possibility that posterior probability of normal mails could be bigger than posterior probability of spam mails. That is to say model will be more prone to classify a mail as normal rather than spam. Therefore, the number of spam mails in results can be highly related to the ratio of the spam mails in the training set. For example, if we have a normal majority training set and spam majority test set, the results can be awful in the prediction stage because of the bias.

2.2 Multinomial Naive Bayes Model

In order to train a Multinomial Naive Bayes Model, the function prediction2_2(x_array, y_array,test_x) is used (see appendix) as seen from the code, I have initialize the parameters in matrix form. In this way I could reach every parameter easily. Also thanks to this initialization I didn't have to use a loop to calculate every other probability. I just directly used numpy and its sum function to perform log likelihood calculations. In this way the run-time of the code becomes much more smaller.

Also as seen from the code, I have a variable which is "small", which is 10^{-12}, I initialized this variable because I got -inf error. With the usage of "small" I got rid of the -inf error.
After trained and tested I got 995 correct prediction and 40 wrong prediction which is 96.1353% accuracy. 26 of wrong predictions were predicted as 1 (spam) while the real data was 0(normal), and 14 of wrong predictions were predicted as 0(normal) while the real data was 1(spam). The full confusion matrix is shown below in figure-1.

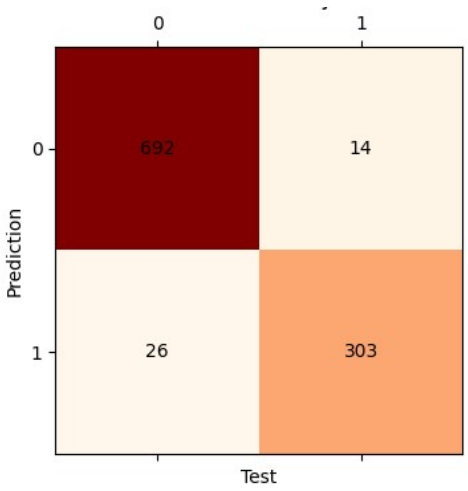

Figure 1: Confusion Matrix in Q2.2

Also I checked the total number of spam mails in test set and it was 317 out of 1035 mails which

corresponds to 30.628%. This ratio for the train set was 28.59%. As seen those ratios are pretty close to each other. Therefore, as mentioned before, the model was more prone to predict a data as normal rather than spam because of prior, and if test set has more spam mails than normal mails the accuracy could be lower than the what I got. Since the ratios of spam mails are close to each other for train and test sets the accuracy of the predictions was high.

2.3 Multinomial Naive Bayes Model with Drichlet Prior

In this part of the homework I have a new thing which is Dirichlet prior α. This is related to additive smoothing, where the prior assumes that each word appears α times in the training set in order to make it fair by "hallucinating." In this part the function I used for 2.2 was used with a small revise to add α, which is coded as prediction2_3(x_array, y_array,test_x,alpha) (see appendix).

After trained and tested I got 983 correct prediction and 52 wrong prediction which is 94.97% accuracy. 39 of wrong predictions were predicted as 1 (spam) while the real data was 0(normal), and 13 of wrong predictions were predicted as 0(normal) while the real data was 1(spam). The full confusion matrix is shown below in figure-2.

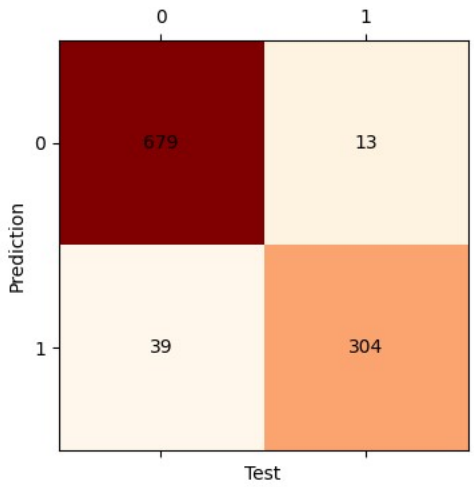

Figure 2: Confusion Matrix in Q2.3, Drichlet prior=5

In this case I have taken α as 5 and it is seen that the accuracy of the prediction has reduced. Normally I was expecting this parameter to make the accuracy higher because when it is added, zero probabilities replaced and in this way those terms don't reduce the overall probability. However, in Q2.2 I had to add a "small" variable so to get rid of zero terms. Also I have seen that adding 5 extra words to mails could make the probability worse. I think every mail doesn't have to contain some words in data set that much. Then I try 10^{-10} as Drichlet prior and get better results.After trained and tested, got 996 correct prediction and 39 wrong prediction which is 96.23% accuracy. 22 of wrong predictions were predicted as 1 (spam) while the real data was 0(normal), and 17 of wrong predictions were predicted as 0(normal) while the real data was 1(spam). The full confusion matrix is shown below in figure-3.

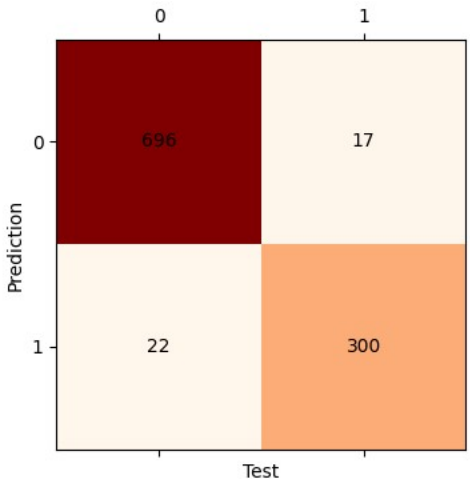

Figure 3: Confusion Matrix in Q2.3, Drichlet prior=10^{-10}

2.4 Bernoulli Naive Bayes Model

In order to train a Bernoulli Naive Bayes Model, the function bernoulli_prediction(train_x, train_y,test_x) is used (see appendix) as seen from the code, I have initialize the parameters in matrix form similar to previous part and calculate the probabilities using those matrices. Also I have used "small" parameter again in order to get rid of -inf error.

After trained and tested I got 951 correct prediction and 84 wrong prediction which is 91.88% accuracy. 27 of wrong predictions were predicted as 1 (spam) while the real data was 0(normal), and 57 of wrong predictions were predicted as 0(normal) while the real data was 1(spam). The full confusion matrix is shown below in figure-4.

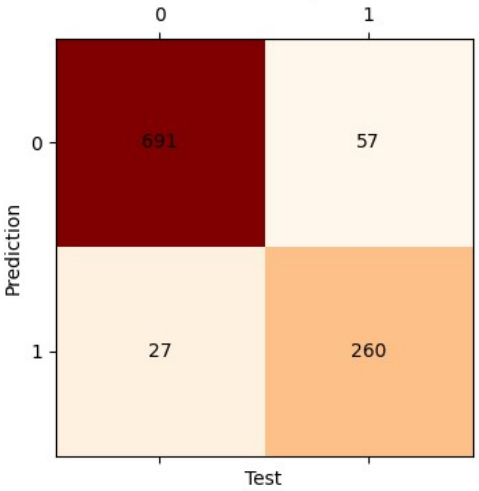

Figure 4: Confusion Matrix in Q2.4

As seen from the bernoulli prediction result the accuracy is smaller than the multinomial prediction result. Therefore it can be said by bernoulli prediction result that existence of certain words are important while classifying an e-mail as spam and normal. Also it can be said by comparing multinomial and bernoulli

prediction results, not only the existence of words are important, but also the number of occurence of the words are important while classifying the mails as spam and normal.

2.5

Considering the confusion matrices that I have obtained during the task, it is seen that Multinomial Naive Bayes Model with a Drichlet Prior(α =10^{-10}) brought the best result. Then it is followed by Multinomial Naive Bayes Model with a small value to get rid of zero terms. The third biggest accuracy came from Naive Bayes Model with a Drichlet Prior(α =5) and finally the lowest accuracy come from Bernoulli Naive Bayes model. Even if all the accuracy ratios are bigger than 90% the models may not be that accurate in real life applications. This is because as I mentioned before the training set has not balanced distribution. Since the training set is skewed toward non-spam e-mails, the models are more prone to classify an e-mail as normal rather than spam. Therefore, those accuracies can be deceiving for real life applications. Even if I may not get such high scores in real life, I believe Multinomial Naive Bayes Model with a Drichlet Prior(α =10^{-10}) would bring the best result because during this task I understand that not only the existence of a word is important, but also word frequency is also significant when you classify an e-mail. On top of those, in this work we have not removed the stop words from the data sets. If they were taken into account with a bigger and balanced data set, the model could give better results.

3 Appendix

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


# In[2]:


###Change root variable###
root = r'C:\Users\Oguz\Desktop\Bilkent\3.2\CS464\hw1'
os.chdir(root)


## <font color='red'>Q2.1</font>
```

```
# In[3]:


def counter(array):
    spam_count=len(np.argwhere(array == 1))
    total_count=len(array)
    ham_count=total_count-spam_count
    return spam_count, ham_count


### <font color='red'>Q2.2</font>

# In[4]:


def prediction2_2(x_array, y_array,test_x):
    T_j_spam= x_array[np.argwhere(y_array == 1).T[0]]
    T_j_normal= x_array[np.argwhere(y_array == 0).T[0]]
    N_normal , N_spam= counter(y_array)
    N = len(y_array)
    Pi_normal= N_normal/N
    Pi_spam=N_spam/N
    small=1e-12
    theta_j_spam= T_j_spam.sum(axis=0) /(T_j_spam.sum())
    theta_j_normal= T_j_normal.sum(axis=0) /(T_j_normal.sum())
    prob_spam= np.log(Pi_spam) + \
    (test_x * np.log(theta_j_spam+small)).sum(axis=1)
    prob_normal=np.log(Pi_normal) + \
    (test_x * np.log(theta_j_normal+small)).sum(axis=1)
    return (prob_spam>prob_normal)*1


### <font color='red'>Q2.3</font>

# In[5]:


def prediction2_3(x_array, y_array,test_x,alpha):
    T_j_spam= x_array[np.argwhere(y_array == 1).T[0]]
    T_j_normal= x_array[np.argwhere(y_array == 0).T[0]]
    N_normal , N_spam= counter(y_array)
    N = len(y_array)
```

```
    Pi_normal= N_normal/N
    Pi_spam=N_spam/N
    theta_j_spam=  (T_j_spam.sum(axis=0)+ alpha) /\
    (T_j_spam.sum() +alpha *len(x_array.T))
    theta_j_normal=  (T_j_normal.sum(axis=0) + alpha) /\
    (T_j_normal.sum() + alpha *len(x_array.T) )
    prob_spam= np.log(Pi_spam) +\
        (test_x * np.log(theta_j_spam)).sum(axis=1)
    prob_normal=np.log(Pi_normal) +\
        (test_x * np.log(theta_j_normal)).sum(axis=1)
    return (prob_spam>prob_normal)*1


### <font color='red'>Q2.3</font>

# In[6]:


def bernoulli_prediction(train_x, train_y,test_x):
    normal_matrix=((train_x!=0)*1)[np.argwhere(train_y == 0).T[0]]
    spam_matrix=((train_x!=0)*1)[np.argwhere(train_y == 1).T[0]]
    S_j_spam=spam_matrix.sum(axis=0)
    S_j_normal=normal_matrix.sum(axis=0)
    N_normal , N_spam= counter(train_y)
    N = len(train_y)
    Pi_normal= N_normal/N
    Pi_spam=N_spam/N
    theta_j_spam_=  S_j_spam   / len(spam_matrix)
    theta_j_normal_=  S_j_normal/ len(normal_matrix)
    test_matrix=(test_x != 0)*1
    small=1e-12
    prob_spam= np.log(Pi_spam) + np.log( test_matrix * theta_j_spam_\
        + (1-test_matrix)* (1- theta_j_spam_)+small).sum(axis=1)
    prob_normal= np.log(Pi_normal) + np.log( test_matrix * theta_j_normal_\
        + (1-test_matrix)* (1- theta_j_normal_)  +small).sum(axis=1)

    return 1*(prob_spam>prob_normal)


# In[7]:


def confusion(test_y,result,ax,title="Result"):
```

```
    correct=len(np.argwhere(test_y==result))
    accuracy= 100*correct/len(test_y)
    confusion = pd.crosstab(test_y, result)
    ax.matshow(confusion,cmap='OrRd')
    ax.set(xlabel='Test', ylabel='Prediction')
    for i in range(2):
        for j in range(2):
            c = confusion[j][i]
            ax.text(i, j, str(c), va='center', ha='center')
    ax.set_title(title)
    print("Accuracy for {} is: {:.3f} %".format(title,accuracy))


# In[8]:


def result(a):
    train_x=pd.read_csv("x_train.csv").to_numpy()
    train_y=pd.read_csv("y_train.csv").to_numpy()
    test_x=pd.read_csv("x_test.csv").to_numpy()
    test_y=pd.read_csv("y_test.csv").to_numpy().T[0]
    fig, ax = plt.subplots()
    if a==1:
        result=prediction2_2(train_x,train_y,test_x)
    elif a==2:
        result=prediction2_3(train_x,train_y,test_x,5)
    elif a==3:
        result=prediction2_3(train_x,train_y,test_x,1e-10)
    else:
        result=bernoulli_prediction(train_x, train_y,test_x)
    confusion(test_y,result,ax)
    plt.show()


# In[9]:


def main():
    train_x=pd.read_csv("x_train.csv").to_numpy()
    train_y=pd.read_csv("y_train.csv").to_numpy()
    test_x=pd.read_csv("x_test.csv").to_numpy()
    test_y=pd.read_csv("y_test.csv").to_numpy().T[0]
    spam_count,ham_count= counter(train_y)
```

```
    spam_percentage= 100 * spam_count/(spam_count+ham_count)
    print("The percentage of spam e-mails in the \
y_train.csv is: {:.3f} %".format(spam_percentage))
    data=np.c_[train_x,train_y]
    np.random.shuffle(data)
    train_x=data[:,:-1]
    train_y=data[:,-1]
    r1=prediction2_2(train_x,train_y,test_x)
    r2=prediction2_3(train_x,train_y,test_x,5)
    r3=prediction2_3(train_x,train_y,test_x,1e-10)
    r4=bernoulli_prediction(train_x, train_y,test_x)
    plt.rcParams["figure.figsize"] = [12,8]
    plt.rcParams["figure.autolayout"] = True
    fig, ax = plt.subplots(2,2)
    plt.title("Confusion Matrices")
    confusion(test_y,r1,ax[0,0],"Multinomial Naive Bayes Model")
    confusion(test_y,r2,ax[0,1],"Multinomial Naive Bayes Model \
with Drichlet Prior 5")
    confusion(test_y,r3,ax[1,0],"Multinomial Naive Bayes Model \
with Drichlet Prior 10^(-10)")
    confusion(test_y,r4,ax[1,1],"Bernoulli Naive Bayes Model")
    plt.show()


# In[11]:


if __name__ == '__main__':
    main()
```