CS 464 – Introduction to Machine Learning

Homework 1

Dora Tütüncü • 21801687 • Section-2

1. The White Library

(1.1)

Sample space contains all possibilities of a particular even happening in an environment. The events of novels, poetry books and story books and the number of words in their titles can be put in a table as seen below.

	novels	poetry books	story books
min title words	1	1	1
max title words	2	3	3

3		(P,3)	(S,3)
2	(N,2)	(P,2)	(S,2)
1	(N,1)	(P,1)	(S,1)
	Novels	**Poetry**	**Stories**

The sample set Ω contains all these events, hence is defined as:

$$\Omega = \{(N,1),(N,2),(P,1),(P,2),(P,3),(S,1),(S,2),(S,3)\}$$

(1.2)

$$A = \{\text{"}A\ novel\ with\ a\ single\ word\ title\ \boldsymbol{OR}\ a\ poem\ with\ a\ 2\ \boldsymbol{or}\ 3\ word\ title, but\ not\ \boldsymbol{NOT}\ a\ story\ book\}$$

$$A = \{(N,1),(P,2),(P,3)\}$$

(1.3)

Axioms of Probability:

1. The probability of an event is greater or equal to zero $\Longrightarrow P(X) \geq 0$
2. The total probability of all events in a sample space is $\Longrightarrow P(\Omega) \geq 0$
3. For any mutually exclusive events, such that $X_Q \cap X_R = \emptyset$, probability is;
   $P\left(\bigcup_Q X_Q\right) = \sum_Q P(X_Q)$

(1.4)

$P(3\ word\ story\ book\ \boldsymbol{OR}\ 2\ word\ novel) = 0.045$
$P(3\ word\ story\ book\ \boldsymbol{OR}\ 2\ word\ novel\ \boldsymbol{OR}\ 2\ word\ poetry) = 0.11$
$P(2\ word\ poetry\ book\ \boldsymbol{OR}\ 3\ word\ story\ book) = 0.06$

Let the event of a 3 word title story book be A
Let the event of a 2 word title novel book be B
Let the event of a 2 word title poetry book be C

These events are independent, and are hence mutually exclusive.
$P(A\ or\ B) = P(A) + P(B) = 0.045$ **(1)**
$P(A\ or\ B\ or\ C) = P(A) + P(B) + P(C) = 0.11$ **(2)**
$P(C\ or\ A) = P(A) + P(C) = 0.06$ **(3)**

Subtracting (2) and (1), we obtain:
$P(C) = 0.065$ **(4)**

Comparing equations (3) and (4), it can be seen that $P(A) + P(C) = 0.06$ is impossible, as due to other conditions, $P(C) = 0.065$ which would imply that $P(A) = -0.005$, which violates the 1st axiom of probability.

$\Longrightarrow$ Hence, Donald's estimation is **incorrect.**

2. Café Customers

(2.1)

If 10 people arrive at the break, what is the possibility **less than** 3 people will buy coffee?

$P(Person\ takes\ Coffee) = P(C) = 0.3\,, \qquad P(C') = 0.7$

$$P(Y < 3) = \binom{10}{0}(0.3)^0(0.7)^{10} + \binom{10}{1}(0.3)^1(0.7)^9 + \binom{10}{2}(0.3)^2(0.7)^8$$

$$= (0.7)^{10} + 10 \cdot (0.3)^1(0.7)^9 + 45 \cdot (0.3)^2(0.7)^8 = \mathbf{0.3828}$$

(2.2)

Probability of 2 people arriving and none of them buy coffee $\Longrightarrow$ A

$$P(A) = P(X = 2) \cdot P(Y = 0) = \left(e^{-20} \cdot \frac{20^2}{2!}\right) \cdot \left[\binom{10}{0}(0.3)^0(0.7)^2\right] = 200(0.49)e^{-20}$$

$$\boldsymbol{P(A) = 2.0199 \cdot 10^{-7}}$$

(2.3)

$E[Y] =?$

Let k be the number of people in Mozart Café during the break.

$$E[Y] = \sum_{i=0}^{k} i \cdot P(Y = i) = \sum_{i=0}^{k} i \cdot \binom{k}{i} (0.3)^i (0.7)^{k-i} = (0.3)k \sum_{i=1}^{k} \frac{(k-1)!}{(i-1)! \cdot (k-i)!} (0.3)^{i-1} (0.7)^{k-i}$$

Let $a = i - 1$, $b = k - 1$, $a + 1 = i$, $b + 1 = k$, $k - i = b - a$

$$E[Y] = (0.3)K \sum_{a=0}^{k} \frac{b!}{a!\,(b-a)!} (0.3)^a (0.7)^{b-a} = (0.3)K \sum_{a=0}^{k} \binom{b}{a} (0.3)^a (0.7)^b$$

$$\boldsymbol{E[Y] = (0.3)K}$$

3. Spam Detection

(3.1)

After reading the y_train.csv file, it is determined that there are 2911 spam files, and a total of 4085 files composing the training dataset. Therefore, around 71% of the training dataset is composed of spam mails, which means the dataset is indeed slightly skewed towards spam. This will negatively affect the results as test documents will be more likely to be interpreted as spam, however since the skew is not too large, there shouldn't be drastic interpretation errors. If it were, for example, skewed to spam mail by 95%, then there would be major issues in the machine learning results. The issue for these datasets could perhaps be removed by removing 868 random spam files to obtain a perfectly balanced dataset, but by doing so it is possible to lose out on important data by undersampling. Another possible way to overcome the skew is to copy some of the normal mails by oversampling. However, these methods seem to be unnecessary for the dataset at hand due to the fact that the ratio of spam and normal mails aren't too skewed, hence it is possible to directly work with this dataset.

(3.2)

After training the Multinomial Naïve Bayes MLE classifier, the following results were obtained:

- Number of True Positives: 611
- Number of True Negatives: 317
- Number of False Positives: 8
- Number of False Negatives: 150
- Accuracy is 85.451197053407 %
- Precision is 98.70759289176091 %
- Total time is 16.1942 seconds

This implies that 928 out of 1086 test documents were labeled correctly, and there were a total of 158 wrongly predicted files, with most of them being false negatives, i.e., spam documents have been classified as normal.

(3.3)

After training the Multinomial Naïve Bayes MAP classifier, the following results were obtained:

- Number of True Positives: 751
- Number of True Negatives: 315
- Number of False Positives: 10
- Number of False Negatives: 10
- Accuracy is 98.15837937384899 %
- Precision is 98.68593955321944 %
- Total time is 17.4232 seconds

This implies that 1066 out of 1086 test documents were labeled correctly, and there were a total of 20 wrongly predicted files, perfectly distributed between false negatives and false positives. Due to smoothing, accuracy was higher as it "hallucinated" that there was at least one word of the vocabulary in every training file, eliminating the negative infinity prior probability results that emerges when calculating the posterior. By doing so, accuracy was significantly improved.

(3.4)

After training the Bernoulli Naïve Bayes classifier, the following results were obtained:

- Number of True Positives: 613
- Number of True Negatives: 300
- Number of False Positives: 25
- Number of False Negatives: 148
- Accuracy is 84.06998158379373 %
- Precision is 96.08150470219435 %
- Total time is 33.8550 seconds

This implies that 913 out of 1086 test documents were labeled correctly, and there were a total of 173 wrongly predicted files, with most of them being false negatives, i.e. spam documents have been classified as normal.

This method gives more or less similar results to the Multinomial Naïve Bayes MLE classifier; however it could not achieve results as good as the MAP classifier. This happens because including, or in other word, "hallucinating" rare words that may or may not happen trains the data so that it has a clearer understanding of what spam and normal mail should be like. Bernoulli also may have given very slightly worse results compared to MLE because it disregarded the amount of usage of a word in a single mail; however in predicting a spam e-mail, the frequency of a word is important.

Code for Question 3

Note: The black highlighted scripts are the respective results of each code. The lines with the # correspond to the coding cell seperations used while writing the program.

```
from google.colab import drive
drive.mount('/content/gdrive')

!ls /content/gdrive/My\ Drive/Dora/Bilkent/CS464/HW1 # Use YOUR OWN DIRECTORY!!
import os
import csv
import math
import random
import operator
import pdb
import time

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import scipy as scp

from timeit import default_timer as timer
from math import log, inf

np.random.seed(123)

tic = time.perf_counter()

countfiles = 0
countspam = 0

root = '/content/gdrive/My Drive/Dora/Bilkent/CS464/HW1'
csv_yTrain = os.path.join(root, 'y_train.csv')
csv_xTrain = os.path.join(root, 'x_train.csv')
csv_yTest = os.path.join(root, 'y_test.csv')
csv_xTest = os.path.join(root, 'x_test.csv')
vocab = os.path.join(root, 'vocabulary.txt')

import numpy as np
import pandas as pd

#read the csv files ot extract the feature and labels from the test and
train_feature = pd.read_csv(csv_xTrain, header=None)
train_label = pd.read_csv(csv_yTrain, header=None)
test_feature = pd.read_csv(csv_xTest, header=None)
test_label = pd.read_csv(csv_yTest, header=None)

labels = ['0' , '1']
vocabcount = 44020 #since this quantity is given, it is directly used, however it could've also been easily calculated.

toc = time.perf_counter()
print(f"\nTotal time to obtain features and labels for the training and test documents is {toc - tic:0.4f} seconds")



################################################################################

tic = time.perf_counter()

#generate the train_multinomial concatanated matrix to assign training features to their respective labels.
train_multinomial = pd.concat((train_label[0].rename('label'), train_feature), axis=1)

#count the number of spam and normal mails in the training dataset
label_counts = train_label[0].value_counts()
normal_mails = label_counts[0]
spam_mails = label_counts[1]
total_mails = normal_mails + spam_mails

#calculate the probabilities of choosing a normal or a spam email from the dataset.
prob_normal = normal_mails/total_mails
prob_spam = spam_mails/total_mails

toc = time.perf_counter()
print(f"\nTotal time to calculate number of mail types and probability of choosing a spam or normal mail out of the dataset i
s {toc - tic:0.4f} seconds")
```

```
###############################################################################

tic = time.perf_counter()

#find the number of occurances of a specific word in the spam and normal datasets, and store it in occurances.
occurances = train_multinomial.groupby('label').sum()

#extract the total number of each word used in the normal and spam mails seperately.

normal_total = occurances.sum(axis=1)[0]
spam_total = occurances.sum(axis=1)[1]

#count the number of times that a particular word has occured. This will be used for the Bernoulli Model.
countoccurance = train_multinomial.mask(train_multinomial > 1, 1).groupby('label').sum()

toc = time.perf_counter()
print(f"\nTotal time to find the number of times a word occurs and the count a word occuring in the training datasets is {toc
 - tic:0.4f} seconds")

###############################################################################

def confusion_results(prediction_matrix):

    #this function is defined to compute the confusion matrix values for the ML models.

    true_positive = 0
    true_negative = 0
    false_negative = 0
    false_positive = 0

    for i in range(len(prediction_matrix)):
        if prediction_matrix[i] == 1 and prediction_matrix[i] == test_label[0][i] :
            true_positive += 1
        elif prediction_matrix[i] == 0 and prediction_matrix[i] == test_label[0][i] :
            true_negative += 1
        elif prediction_matrix[i] == 1 and prediction_matrix[i] != test_label[0][i] :
            false_positive += 1
        elif prediction_matrix[i] == 0 and prediction_matrix[i] != test_label[0][i] :
            false_negative += 1

    accuracy = (true_positive+true_negative)/(true_negative+true_positive+false_negative+false_positive)*100
    precision = true_positive / (true_positive + false_positive)*100

    print('Number of True Positives: ', true_positive)
    print('Number of True Negatives: ', true_negative)
    print('Number of False Positives: ', false_positive)
    print('Number of False Negatives: ', false_negative)
    print('Accuracy is ', accuracy, ' %')
    print('Precision is ', precision, ' %')


###############################################################################

def MLE_predictor(email):

    #this function is used to calculate the posteriors from the Multinomial MLE model, given its priors.
    #the posteriror for spam and normal will be calculated and compared to reach to a prediction for a particular email docum
ent.

    yi_normaldf = (email * np.log((occurances.iloc[0])/(normal_total))).fillna(0)
    yi_spamdf = (email * np.log((occurances.iloc[1])/(spam_total))).fillna(0)

    yi_normal = yi_normaldf.values.sum() + math.log(prob_normal)
    yi_spam = yi_spamdf.values.sum() + math.log(prob_spam)



    if yi_normal > yi_spam:
        prediction = 0
    elif yi_normal < yi_spam:
        prediction = 1
    elif yi_normal == yi_spam:
        prediction = 0

    return prediction
```

```
#############################################################################

def MAP_predictor(email):

    #this function is used to calculate the posteriors from the MAP model, given its priors.
    #the posteriror for spam and normal will be calculated and compared to reach to a prediction for a particular email docum
ent.

    yi_normaldf = (email * np.log((occurances.iloc[0]+1)/(normal_total+44020))).fillna(0)
    yi_spamdf = (email * np.log((occurances.iloc[1]+1)/(spam_total+44020))).fillna(0)

    yi_normal = yi_normaldf.values.sum() + math.log(prob_normal)
    yi_spam = yi_spamdf.values.sum() + math.log(prob_spam)



    if yi_normal > yi_spam:
        prediction = 0
    elif yi_normal < yi_spam:
        prediction = 1
    elif yi_normal == yi_spam:
        prediction = 0

    return prediction




#############################################################################

def Bernoulli_predictor(email):

    #this function is used to calculate the posteriors from the Bernoulli model, given its priors.
    #the posteriror for spam and normal will be calculated and compared to reach to a prediction for a particular email docum
ent.

    newmail = email.mask(email > 1, 1)
    compnewmail = 1-newmail

    yi_normaldf = (newmail * np.log((countoccurance.iloc[0])/normal_mails)).fillna(0) + ((compnewmail)*(np.log(1-
(countoccurance.iloc[0])/normal_mails))).fillna(0)
    yi_spamdf = (newmail * np.log((countoccurance.iloc[1])/(spam_mails))).fillna(0) + ((compnewmail)*(np.log(1-
(countoccurance.iloc[1])/spam_mails))).fillna(0)

    yi_normal = yi_normaldf.values.sum() + math.log(prob_normal)
    yi_spam = yi_spamdf.values.sum() + math.log(prob_spam)

    if yi_normal > yi_spam:
        prediction = 0
    elif yi_normal < yi_spam:
        prediction = 1
    elif yi_normal == yi_spam:
        prediction = 0

    return prediction


#############################################################################

prediction_matrix_MLE = []
np.warnings.filterwarnings('ignore')

tic = time.perf_counter()

#the prediction for every document will be conducted from the test features, using the corresponding model.
#predictions will be stored in its correspondig matrix, each index referring to the respective document.

for i in range(len(test_label)):
    prediction_matrix_MLE.append(MLE_predictor(test_feature.iloc[i]))

print("\nThe results for MLE classification are:")
confusion_results(prediction_matrix_MLE)

toc = time.perf_counter()
print(f"Total time for the MLE classification is {toc - tic:0.4f} seconds")
```

```
#############################################################################

prediction_matrix_MAP = []

tic = time.perf_counter()

#the prediction for every document will be conducted from the test features, using the corresponding model.
#predictions will be stored in its correspondig matrix, each index referring to the respective document.

for i in range(len(test_label)):
    prediction_matrix_MAP.append(MAP_predictor(test_feature.iloc[i]))

print("\nThe results for MAP classification are:")
confusion_results(prediction_matrix_MAP)

toc = time.perf_counter()
print(f"Total time for the MAP classification is {toc - tic:0.4f} seconds")


#############################################################################

prediction_matrix_Bernoulli = []
np.warnings.filterwarnings('ignore')

tic = time.perf_counter()

#the prediction for every document will be conducted from the test features, using the corresponding model.
#predictions will be stored in its correspondig matrix, each index referring to the respective document.

for i in range(len(test_label)):
    prediction_matrix_Bernoulli.append(Bernoulli_predictor(test_feature.iloc[i]))

print("\nThe results for Bernoulli classification are:")
confusion_results(prediction_matrix_Bernoulli)

toc = time.perf_counter()
print(f"Total time for the MAP classification is {toc - tic:0.4f} seconds")
```

Results:

```
Total time to obtain features and labels for the training and test documents is 155.4182 seconds

Total time to calculate number of mail types and probability of choosing a spam or normal mail out of the dataset is 1.1472
seconds

Total time to find the number of times a word occurs and the count a word occuring in the training datasets is 19.1628
seconds

The results for MLE classification are:
Number of True Positives:  611
Number of True Negatives:  317
Number of False Positives:  8
Number of False Negatives:  150
Accuracy is  85.451197053407  %
Precision is  98.70759289176091  %
Total time for the MLE classification is 16.4976 seconds

The results for MAP classification are:
Number of True Positives:  751
Number of True Negatives:  315
Number of False Positives:  10
Number of False Negatives:  10
Accuracy is  98.15837937384899  %
Precision is  98.68593955321944  %
Total time for the MAP classification is 17.4877 seconds

The results for Bernoulli classification are:
Number of True Positives:  613
Number of True Negatives:  300
Number of False Positives:  25
Number of False Negatives:  148
Accuracy is  84.06998158379373  %
Precision is  96.08150470219435  %
Total time for the MAP classification is 34.3000 seconds
```