Mehmet Bayık
21802166
CS464-01
12.11.2022

HOMEWORK-1 REPORT

Question 1) Probability Review [10 pts]

There are 2 boxes in a room. The first box contains 2 blue coins and 1 yellow coin. The second box contains 1 blue and 1 red coin. The blue coins are fair. However, the yellow coin has 25% and red coin has 10% chance of landing heads. You randomly select a coin from one of the boxes and toss it two times.

First Box: 2 Blue, 1 Yellow // **Second Box:** 1 Blue, 1 Red

$$P(Heads \mid Coin = Blue) = 0.5$$
$$P(Heads \mid Coin = Yellow) = 0.25$$
$$P(Heads \mid Coin = Red) = 0.1$$

Q 1.1) $\boldsymbol{P(Two\ Heads\ in\ a\ row)} = ?$

$$\begin{aligned} P(HH) &= P(Blue, HH) + P(Yellow, HH) + P(Red, HH) \\ &= P(First\ Box) \cdot P(\ Blue|First\ Box) \cdot [P(Heads \mid Coin = Blue)]^2 \\ &+ P(Second\ Box) \cdot P(\ Blue|Second\ Box) \cdot [P(Heads \mid Coin = Blue)]^2 \\ &+ P(First\ Box) \cdot P(\ Yellow|First\ Box) \cdot [P(Heads \mid Coin = Yellow)]^2 \\ &+ P(Second\ Box) \cdot P(\ Red|Second\ Box) \cdot [P(Heads \mid Coin = Red)]^2 \end{aligned}$$

$$P(HH) = \left(\frac{1}{2}\right)\left(\frac{2}{3}\right)(0.5)^2 + \left(\frac{1}{2}\right)\left(\frac{1}{2}\right)(0.5)^2 + \left(\frac{1}{2}\right)\left(\frac{1}{3}\right)(0.25)^2 + \left(\frac{1}{2}\right)\left(\frac{1}{2}\right)(0.1)^2$$

$$P(HH) = \mathbf{0.15875}$$

Q 1.2) $\boldsymbol{P(\ Blue\ |Two\ Heads\ in\ a\ row)} = ?$

$$P(\ Blue\ |HH) = \frac{P(Blue, HH)}{P(HH)} = \frac{P(Blue, HH)}{P(HH)} = \frac{0.14583}{0.15875} = \mathbf{0.91861}$$

Q 1.3) $\boldsymbol{P(\ Red\ |Two\ Heads\ in\ a\ row)} = ?$

$$P(\ Red\ |HH) = \frac{P(Red, HH)}{P(HH)} = \frac{0.00250}{0.15875} = \mathbf{0.01574}$$

Mehmet Bayık
21802166
CS464-01
12.11.2022

Question 2) MLE and MAP [20 pts]

Suppose you have n data points x1,x2....xn. After visualizing your data, you think that these data points are coming from a normal distribution. The probability density function for the normal distribution is given as:

$$f(x;\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

Q 2.1) Find MLE for the μ using given data points.

Log means natural logarithm ln for the sake of this solution.

$$L(p) = \prod_{i=1}^{n} P(X = x_i)$$

$$L(X|\mu) = \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} e^{-(x_i-\mu)^2/2\sigma^2}$$

$$\log\big(L(X|\mu)\big) = \sum_{i=1}^{n} -\log\big(\sigma\sqrt{2\pi}\big) - \frac{(x_i-\mu)^2}{2\sigma^2}$$

$$\log\big(L(X|\mu)\big) = -\,\mathrm{n}\cdot\log\big(\sigma\sqrt{2\pi}\big) - \sum_{i=1}^{n} -\frac{(x_i-\mu)^2}{2\sigma^2}$$

$$\frac{d\log\big(L(X|\mu)\big)}{d\mu} = \frac{1}{\sigma^2}\sum_{i=1}^{n}(x_i-\mu) = 0$$

$$MLE\ \hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Q 2.2) Suppose Prior of μ is exponential distribution with parameter λ. Find MAP estimate of μ.

$$f(x,\mu) = \begin{matrix} \lambda e^{-\lambda\mu} & ,for\ \mu \geq 0 \\ 0 & ,otherwise \end{matrix}$$

According to Bayes Rule:

$$MAP = P(\mu|X) \propto P(X|\mu)\cdot P(\mu)$$

We already know Likelihood. Just multiply with prior to get posterior.

Mehmet Bayık
21802166
CS464-01
12.11.2022

$$\frac{d\ \log\big(L(\mu|X)\big)}{d\mu} = \frac{d\log\big(L(X|\mu)\big)}{d\mu} + \frac{d\log(L(\mu))}{d\mu} = \frac{1}{\sigma^2}\sum_{i=1}^{n}(x_i - \mu) + \frac{d\log(L(\mu))}{d\mu}$$

$$= \frac{1}{\sigma^2}\sum_{i=1}^{n}(x_i - \mu) + \frac{d}{d\mu}\big(\log\left(\lambda e^{-\lambda\mu}\right)\big)$$

$$= \frac{1}{\sigma^2}\sum_{i=1}^{n}(x_i - \mu) + \frac{d}{d\mu}(-\lambda^2\mu)$$

$$0 = \frac{1}{\sigma^2}\sum_{i=1}^{n}(x_i - \mu)\ - \lambda^2$$

$$MAP\ \hat{\mu} = -\lambda^2 + \frac{1}{n}\sum_{i=1}^{n}x_i$$

Q 2.3) **$\mu = 1, \sigma^2 = 1$**

$$P(x_{n+1} = 1) = ?$$

$$f(x) = \frac{1}{\sqrt{2\pi}}e^{-(x-1)^2/2}$$

$$P(x_{n+1} = 1 \mid X\ ) = \frac{1}{\sqrt{2\pi}} = \mathbf{0.3989}$$

What is the likelihood of the data point xn+1 = 2 according to pdf of normal distribution? We know that $x_{n+1} = 2$.

$$P(x_{n+1} = 2 \mid X) = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}} = \mathbf{0.2419}$$

$$P(X \mid x_{n+1} = 2) = 2 \cdot \mathbf{0.2419} = \mathbf{0.4838}$$

Mehmet Bayık
21802166
CS464-01
12.11.2022

Question 3) BBC News Classification [70 pts]

In this question, we are given with a dataset which is already preprocessed. So, we do not need to split this data. I will use this dataset as it is.

Q 3.1) Answer following questions.

3.1.1. Class distribution is given below

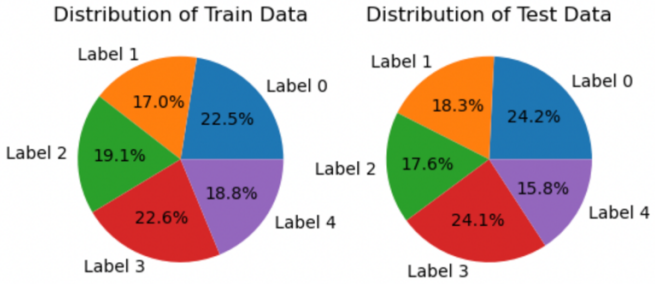

Fig.1: Pie Chart of Class Distribution of Training and Test Data.

3.1.2. Prior Probability of each class given below.

```
Prior Probability of Class 0 is: 0.22482014388489208
Prior Probability of Class 1 is: 0.17026378896882494
Prior Probability of Class 2 is: 0.19124700239808154
Prior Probability of Class 3 is: 0.2260191846522782
Prior Probability of Class 4 is: 0.18764988009592326
```

Fig.2: Prior Probability of each class

3.1.3. Training Set looks very balanced, not skewed to any class. So, our model will not be biased towards any class. If training data was skewed to any class, model will be biased and it will predict that class more. Luckily we don't have that problem. Our results will be good, I guess.

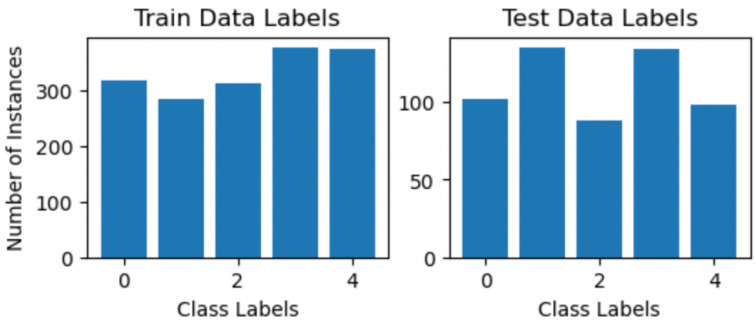

Fig.3: Distribution of Labels in Training and Validation Datasets

3.1.4. Results given below.

P(Alien | Tech) = $3.76 * 10^{-5}$

P(Thunder | Tech) = 0

```
Number of alien words in tech label:  3
Number of thunder words in tech label:  0
Number of all words in tech label:  79685


Probability of alien words in tech label 3.764823994478258e-05
Probability of thunder words in tech label 0.0


Log Probability of alien words in tech label -10.187224352625625
Log Probability of thunder words in tech label -inf
```

Fig.4: Results for Part 3.1.4

Q 3.2) Multinomial Naïve Bayes MLE Estimation

In this part of homework, we are using Multinomial Naïve Bayes rule with MLE estimation. In Naïve Bayes approach, we use Bayes Rule and assume that all prior probabilities for any feature are independent from each other. In this approach, denominator gives same result for any label, then we can ignore it as suggested in homework instructions. In this approach many word does not exist in many documents and this causes zero probability, which decreases our accuracy significantly. After implementing machine learning model, I got **24.2%** accuracy value for this part, exactly same as given in the lab description as expected result. In code, I didn't defined (-infinity)as any number or function, since it will be work as regularization.

```
Accuracy -> 0.2423698384201077
False predictions -> 422
Correct predictions -> 135
```

Fig.5: Evaluation of Test Data for Multinomial Naïve Bayes MLE Estimator

```
[[135.   0.   0.   0.   0.]
 [102.   0.   0.   0.   0.]
 [ 98.   0.   0.   0.   0.]
 [134.   0.   0.   0.   0.]
 [ 88.   0.   0.   0.   0.]]
```

Fig.6: Confusion Matrix for Multinomial Naïve Bayes MLE Estimator

Mehmet Bayık
21802166
CS464-01
12.11.2022

Q 3.3) MAP with Dirichlet Prior (for Multinomial Naïve Bayes)

In this part of homework, we are using MAP estimation with Dirichlet Prior. Which is basically adding one word for every words. In MLE estimation, many word does not exist in many documents and this causes zero probability. So, this eliminates many data. Since we add one word for every word, we don't eliminate our data this way. So, adding same value does not affect output result much, then we can neglect it. Hence, this model gave **97.7%** accuracy, exactly same as given in the lab description as expected result.

```
Accuracy -> 0.9766606822262118
False predictions -> 13
Correct predictions -> 544
```

Fig.7: Evaluation of Test Data for MAP Estimator with Dirichlet Prior (alpha=1)

```
[[131.   0.   2.   0.   2.]
 [  0.  97.   0.   0.   5.]
 [  1.   0.  96.   0.   1.]
 [  0.   0.   1. 133.   0.]
 [  1.   0.   0.   0.  87.]]
```

Fig.8: Confusion Matrix for MAP Estimator with Dirichlet Prior (alpha=1)

Q 3.4) Bernoulli Naïve Bayes MLE Estimation

In this part of homework, we are using Bernoulli Naïve Bayes rule with MLE estimation. Difference between Multinomial Naïve Bayes, we don't take number of occurrences for every work. We just take 1 if it exists in document, and 0 if it does not exist. Also for smoothing we added alpha = 1 to nominator and added 2*alpha to denominator. After implementing machine learning model, I got **24.2%** accuracy value for this part, exactly same as given in the lab description as expected result.

```
Accuracy -> 0.9658886894075404
False predictions -> 19
Correct predictions -> 538
```

Fig.9: Evaluation of Test Data for MAP Estimator with Dirichlet Prior (alpha=1)

```
[[131.   1.   1.   0.   2.]
 [  1.  95.   0.   0.   6.]
 [  3.   0.  95.   0.   0.]
 [  1.   0.   1. 132.   0.]
 [  2.   1.   0.   0.  85.]]
```

Fig.10: Confusion Matrix for MAP Estimator with Dirichlet Prior (alpha=1)

Comparison

In MLE estimation, many word does not exist in many documents and this causes zero probability. So, this eliminates many data. When we take logarithm, it goes -infinity. Since our argmax() function chooses first occurrence in case of ties, our model did many wrong prediction and most of them were label1. However, in MAP estimation with Dirichlet Prior we can eliminate zero probabilities by changing them with very little value our model become very stable and correct according to validation data. Bernoulli is good but since it does not take occurrence time of words, (it is binary), results was not good as Multinomial Naïve Bayes with Dirichlet Prior.

As given in instructions, run time of my code after loading dataset from file is 2.16 seconds which is less than 5 seconds.

```
Time taken to run the program:  2.160900115966797 seconds
```

Fig.11: Run time of code

Appendix:

```
#importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time

X_train = pd.read_csv('dataset/X_train.csv',delim_whitespace=True)
y_train = pd.read_csv('dataset/y_train.csv', names=['class_label'])
X_test = pd.read_csv('dataset/X_test.csv',delim_whitespace=True)
y_test = pd.read_csv('dataset/y_test.csv', names=['class_label'])

start_time = time.time()

print('The shape of X_train is:',X_train.shape)
print('The shape of y_train is:',y_train.shape)
print('The shape of X_test is:',X_test.shape)
print('The shape of y_test is:',y_test.shape)

train_data = pd.concat([X_train, y_train], axis=1)
test_data = pd.concat([X_test, y_test], axis=1)
```

```
print('The shape of train_data is:',train_data.shape)
print('The shape of test_data is:',test_data.shape)

#class label counts
train_list = list(train_data.class_label.value_counts().sort_index())
test_list = list(test_data.class_label.value_counts().sort_index())
print(train_list)
print(test_list)

# Get the class labels and their counts for train data
train_class_labels = train_data.class_label.unique()
train_class_counts = train_data.class_label.value_counts().sort_index()

# Get the class labels and their counts for test data
test_class_labels = test_data.class_label.unique()
test_class_counts = test_data.class_label.value_counts().sort_index()

# Create subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(6, 2))

# Plot train data on first subplot
ax1.bar(train_class_labels, train_class_counts)
ax1.set_xlabel('Class Labels')
ax1.set_ylabel('Number of Instances')
ax1.set_title('Train Data Labels')

# Plot test data on second subplot
ax2.bar(test_class_labels, test_class_counts)
ax2.set_xlabel('Class Labels')
ax2.set_title('Test Data Labels')

# Show the plot
plt.show(block=False)
plt.pause(0.001)

print('In Training Data')
for i in range(5):
    print("The Percentage of Documents belongs to Label",i,"is:
",round(train_list[i]/(sum(train_list))*10000)/100,"%")

print("\n")
print('In Validation Data')
for i in range(5):
    print("The Percentage of Documents belongs to Label",i,"is:
",round(test_list[i]/(sum(test_list))*10000)/100,"%")
```

```
# Get the class labels and their counts for train data
train_class_labels = ['Label 0', 'Label 1', 'Label 2', 'Label 3', 'Label 4']
train_class_counts = train_list

# Get the class labels and their counts for test data
test_class_labels = ['Label 0', 'Label 1', 'Label 2', 'Label 3', 'Label 4']
test_class_counts = test_list

# Create subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(6, 3))

# Plot train data on first subplot
ax1.pie(train_class_counts, labels=train_class_labels, autopct='%1.1f%%')
ax1.set_title('Distribution of Train Data')

# Plot test data on second subplot
ax2.pie(test_class_counts, labels=test_class_labels, autopct='%1.1f%%')
ax2.set_title('Distribution of Test Data')

# Show the plot
plt.show(block=False)
plt.pause(0.001)

#plotting train and test data distribution to check if they are similar
fig, ax = plt.subplots(1, 1)
bins = np.linspace(train_data.iloc[:,-1].values.min() - .25, train_data.iloc[:,-
1].values.max() + .25, 10)
ax.hist(train_data.iloc[:,-1].values,bins=bins)
ax.hist(test_data.iloc[:,-1].values,bins=bins)
ax.set_title("Data Distributions of Train and Test Data (Blue: Train, Orange: Test)")
ax.set_xlabel('Class Labels')
ax.set_ylabel('Number of Instances')
plt.show(block=False)
plt.pause(0.001)


#splitting data into features and labels for train and test data as numpy arrays
x_train = train_data.iloc[:,:-1].values
x_test = test_data.iloc[:,:-1].values
y_train = train_data.iloc[:,-1].values
y_test =  test_data.iloc[:,-1].values

#splitting data into features and labels for train and test data as pandas dataframes
X_train = train_data.drop('class_label', axis=1)
X_test = test_data.drop('class_label', axis=1)
Y_train = train_data['class_label']
```

```
Y_test =  test_data['class_label']

print("x_train shape: "+ str(np.shape(x_train)))
print("x_test shape: "+ str(np.shape(x_test)))
print("y_train shape: "+ str(np.shape(y_train)))
print("y_test shape: "+ str(np.shape(y_test)))

summ_alien=0
summ_thunder=0
all_tech_words=0
for row_index in range(len(train_data['alien'])):
    if train_data['class_label'][row_index] == 4:
        all_tech_words += train_data.iloc[row_index].sum()
        summ_alien += train_data['alien'][row_index] +
train_data['thunder'][row_index]
        summ_thunder += train_data['thunder'][row_index]
        if train_data['alien'][row_index] + train_data['thunder'][row_index] > 0:
            print(row_index)

print("\n")
print('Number of alien words in tech label: ',summ_alien)
print('Number of thunder words in tech label: ',summ_thunder)
print('Number of all words in tech label: ',all_tech_words)

prob_alien = summ_alien/all_tech_words
prob_thunder = summ_thunder/all_tech_words
log_prob_alien = np.log(prob_alien)
log_prob_thunder = np.log(prob_thunder)

print("\n")
print('Probability of alien words in tech label',prob_alien)
print('Probability of thunder words in tech label',prob_thunder)
print("\n")
print('Log Probability of alien words in tech label',log_prob_alien)
print('Log Probability of thunder words in tech label',log_prob_thunder)

#function to calculate confusion matrix
def comp_confmat(actual, predicted):
    classes = np.unique(actual)
    confusion_matrix = np.zeros((len(classes), len(classes)))
    for i in range(len(classes)):
        for j in range(len(classes)):
           confusion_matrix[i, j] = np.sum((actual == classes[i]) & (predicted ==
classes[j]))
    return confusion_matrix
```

```
train = pd.concat([X_train, Y_train], axis=1)

#finding priors
prior0 = train.loc[train['class_label'] ==
0].select_dtypes(np.number).sum().rename('total0').drop('class_label', axis=0)
prior1 = train.loc[train['class_label'] ==
1].select_dtypes(np.number).sum().rename('total1').drop('class_label', axis=0)
prior2 = train.loc[train['class_label'] ==
2].select_dtypes(np.number).sum().rename('total2').drop('class_label', axis=0)
prior3 = train.loc[train['class_label'] ==
3].select_dtypes(np.number).sum().rename('total3').drop('class_label', axis=0)
prior4 = train.loc[train['class_label'] ==
4].select_dtypes(np.number).sum().rename('total4').drop('class_label', axis=0)
total_prior = X_train.select_dtypes(np.number).sum().rename('total_prior')

#finding total word count in each class
total_word_cnt = np.zeros(5)
for k in range (0,5):
    total_word_cnt[k] = train.loc[train['class_label'] ==
k].select_dtypes(np.number).values.sum()

#finding prior probabilities for each class
prior_prob =
list(train_data.class_label.value_counts().sort_index()/train_data.shape[0])
prior_prob_ln = np.log(prior_prob)
print(total_word_cnt)
for i in range(0,5):
    print('Prior Probability of Class '+str(i)+' is: '+str(prior_prob[i]))


#finding prior probabilities for each class and label as list of lists
minus_inf = -np.inf
prior0_prb = np.zeros(prior0.shape[0])
for i in range(0, prior0.shape[0]):
    if prior0[i] == 0:
        prior0_prb[i] = minus_inf
    else:
        prior0_prb[i] = np.log((prior0[i])/(total_word_cnt[0]))
prior1_prb = np.zeros(prior1.shape[0])
for i in range(0, prior1.shape[0]):
    if prior1[i] == 0:
        prior1_prb[i] = minus_inf
    else:
        prior1_prb[i] = np.log((prior1[i])/(total_word_cnt[1]))
prior2_prb = np.zeros(prior2.shape[0])
for i in range(0, prior2.shape[0]):
```

```
    if prior2[i] == 0:
        prior2_prb[i] = minus_inf
    else:
        prior2_prb[i] = np.log((prior2[i])/(total_word_cnt[2]))
prior3_prb = np.zeros(prior3.shape[0])
for i in range(0, prior3.shape[0]):
    if prior3[i] == 0:
        prior3_prb[i] = minus_inf
    else:
        prior3_prb[i] = np.log((prior3[i])/(total_word_cnt[3]))
prior4_prb = np.zeros(prior4.shape[0])
for i in range(0, prior4.shape[0]):
    if prior4[i] == 0:
        prior4_prb[i] = minus_inf
    else:
        prior4_prb[i] = np.log((prior4[i])/(total_word_cnt[4]))

#making mle predictions
mle_test_prediction = [[None for y in range(5)] for x in range(Y_test.shape[0])]
count_y_0 = 0
count_x = 0

for i in range(X_test.shape[0]):
    row = X_test.iloc[i]
    temp0 = prior0_prb * row.values
    temp0 = prior_prob_ln[0] + np.sum(temp0)
    temp1 = prior1_prb * row.values
    temp1 = prior_prob_ln[1] + np.sum(temp1)
    temp2 = prior2_prb * row.values
    temp2 = prior_prob_ln[2] + np.sum(temp2)
    temp3 = prior3_prb * row.values
    temp3 = prior_prob_ln[3] + np.sum(temp3)
    temp4 = prior4_prb * row.values
    temp4 = prior_prob_ln[4] + np.sum(temp4)

    mle_test_prediction[count_x][0] = temp0
    mle_test_prediction[count_x][1] = temp1
    mle_test_prediction[count_x][2] = temp2
    mle_test_prediction[count_x][3] = temp3
    mle_test_prediction[count_x][4] = temp4

    count_x += 1

#choosing the class with highest probability
mle_test_predict = np.zeros(len(mle_test_prediction))
for k in range(len(mle_test_prediction)):
```

```
    #In case of multiple occurrences of the maximum values,
    # the indices corresponding to the first occurrence are returned.
    mle_test_predict[k] = np.argmax(mle_test_prediction[k])

#calculating accuracy and confusion matrix
correct_predicts = np.sum(np.equal(y_test, mle_test_predict))
acc = correct_predicts / len(y_test)
neg_predicted_value = len(y_test) - correct_predicts

print('\nFOR MULTINOMIAL NAIVE BAYES MLE ESTIMATION')
print("Accuracy -> " + str(acc))
print("False predictions -> " + str(len(y_test) - correct_predicts))
print("Correct predictions -> " + str(correct_predicts))
print(comp_confmat(y_test, mle_test_predict))

#finding prior probabilities for each class and label as list of lists
#using drichlet smoothing alpha = 1
#other values are same as mle part
alpha = 1
prior0_prb_map = np.log((prior0+alpha)/(total_word_cnt[0]+(alpha*X_train.shape[1])))
prior1_prb_map = np.log((prior1+alpha)/(total_word_cnt[1]+(alpha*X_train.shape[1])))
prior2_prb_map = np.log((prior2+alpha)/(total_word_cnt[2]+(alpha*X_train.shape[1])))
prior3_prb_map = np.log((prior3+alpha)/(total_word_cnt[3]+(alpha*X_train.shape[1])))
prior4_prb_map = np.log((prior4+alpha)/(total_word_cnt[4]+(alpha*X_train.shape[1])))

#making map predictions
map_test_prediction = [[None for y in range(5)] for x in range(Y_test.shape[0])]
count_y_0 = 0
count_x = 0

for i in range(X_test.shape[0]):
    row = X_test.iloc[i]
    temp0 = prior0_prb_map * row.values
    temp0 = prior_prob_ln[0] + np.sum(temp0)
    temp1 = prior1_prb_map * row.values
    temp1 = prior_prob_ln[1] + np.sum(temp1)
    temp2 = prior2_prb_map * row.values
    temp2 = prior_prob_ln[2] + np.sum(temp2)
    temp3 = prior3_prb_map * row.values
    temp3 = prior_prob_ln[3] + np.sum(temp3)
    temp4 = prior4_prb_map * row.values
    temp4 = prior_prob_ln[4] + np.sum(temp4)

    map_test_prediction[count_x][0] = temp0
    map_test_prediction[count_x][1] = temp1
    map_test_prediction[count_x][2] = temp2
```

```
    map_test_prediction[count_x][3] = temp3
    map_test_prediction[count_x][4] = temp4

    count_x += 1

#choosing the class with highest probability
map_test_predict = np.zeros(len(map_test_prediction))
for l in range(len(map_test_prediction)):
    map_test_predict[l] = np.argmax(map_test_prediction[l])

#calculating accuracy and confusion matrix
correct_predicts2 = np.sum(np.equal(y_test, map_test_predict))
acc2 = correct_predicts2 / len(y_test)
neg_predicted_value2 = len(y_test) - correct_predicts2

print('\nFOR MULTINOMIAL NAIVE BAYES MAP ESTIMATION')
print("Accuracy -> " + str(acc2))
print("False predictions -> " + str(neg_predicted_value2))
print("Correct predictions -> " + str(correct_predicts2))
print(comp_confmat(y_test, map_test_predict))


#BERNOULLI NAIVE BAYES
X_train_ber = X_train.copy()
X_train_ber[X_train_ber != 0] = 1
train_ber = pd.concat([X_train_ber, Y_train], axis=1)
#finding priors
prior0_ber = train_ber.loc[train_ber['class_label'] ==
0].select_dtypes(np.number).sum().rename('total0').drop('class_label', axis=0)
prior1_ber = train_ber.loc[train_ber['class_label'] ==
1].select_dtypes(np.number).sum().rename('total1').drop('class_label', axis=0)
prior2_ber = train_ber.loc[train_ber['class_label'] ==
2].select_dtypes(np.number).sum().rename('total2').drop('class_label', axis=0)
prior3_ber = train_ber.loc[train_ber['class_label'] ==
3].select_dtypes(np.number).sum().rename('total3').drop('class_label', axis=0)
prior4_ber = train_ber.loc[train_ber['class_label'] ==
4].select_dtypes(np.number).sum().rename('total4').drop('class_label', axis=0)
total_prior_ber = X_train_ber.select_dtypes(np.number).sum().rename('total_prior')

#finding total word count in each class
total_word_cnt_ber = np.zeros(5)
prior_prob_ber = np.zeros(5)
for k in range (0,5):
    total_word_cnt_ber[k] = train_ber.loc[train_ber['class_label'] ==
k].select_dtypes(np.number).values.sum()
```

```
    prior_prob_ber[k] = train_ber.loc[train_ber['class_label'] == k].shape[0] /
X_train_ber.shape[0]


#finding prior probabilities for each class
prior_prob_ln_ber = np.log(prior_prob_ber)
print(prior_prob_ber)
print(total_word_cnt_ber)

X_test_ber = X_test.copy()
X_test_ber[X_test_ber != 0] = 1
test_ber = pd.concat([X_test_ber, Y_test], axis=1)

alpha = 1
prior0_prb_ber = np.log((prior0_ber+alpha)/(total_word_cnt_ber[0]+(alpha*2)))
prior1_prb_ber = np.log((prior1_ber+alpha)/(total_word_cnt_ber[1]+(alpha*2)))
prior2_prb_ber = np.log((prior2_ber+alpha)/(total_word_cnt_ber[2]+(alpha*2)))
prior3_prb_ber = np.log((prior3_ber+alpha)/(total_word_cnt_ber[3]+(alpha*2)))
prior4_prb_ber = np.log((prior4_ber+alpha)/(total_word_cnt_ber[4]+(alpha*2)))

#making bernouilli predictions
ber_test_prediction = [[None for y in range(5)] for x in range(Y_test.shape[0])]
count_y_0 = 0
count_x = 0

for i in range(X_test_ber.shape[0]):
    row = X_test_ber.iloc[i]
    temp0 = prior0_prb_ber * row.values
    temp0 = prior_prob_ln_ber[0] + np.sum(temp0)
    temp1 = prior1_prb_ber * row.values
    temp1 = prior_prob_ln_ber[1] + np.sum(temp1)
    temp2 = prior2_prb_ber * row.values
    temp2 = prior_prob_ln_ber[2] + np.sum(temp2)
    temp3 = prior3_prb_ber * row.values
    temp3 = prior_prob_ln_ber[3] + np.sum(temp3)
    temp4 = prior4_prb_ber * row.values
    temp4 = prior_prob_ln_ber[4] + np.sum(temp4)

    ber_test_prediction[count_x][0] = temp0
    ber_test_prediction[count_x][1] = temp1
    ber_test_prediction[count_x][2] = temp2
    ber_test_prediction[count_x][3] = temp3
    ber_test_prediction[count_x][4] = temp4

    count_x += 1
```

```
#choosing the class with highest probability
ber_test_predict = np.zeros(len(ber_test_prediction))
for l in range(len(ber_test_prediction)):
    ber_test_predict[l] = np.argmax(ber_test_prediction[l])

#calculating accuracy and confusion matrix
correct_predicts3 = np.sum(np.equal(y_test, ber_test_predict)) # CHECKED, matches with
sklearn
acc3 = correct_predicts3 / len(y_test) # CHECKED, matches with sklearn
neg_predicted_value3 = len(y_test) - correct_predicts3 # CHECKED, matches with sklearn

print('\nFOR BERNOULLI NAIVE BAYES')
print("Accuracy -> " + str(acc3))
print("False predictions -> " + str(neg_predicted_value3))
print("Correct predictions -> " + str(correct_predicts3))
print(comp_confmat(y_test, ber_test_predict))

end_time = time.time()
print('\nTime taken to run the program: ',end_time-start_time,'seconds\n')

plt.show()
```