Ayhan Okuyan
21601531
CS464-1

CS464 Introduction to Machine Learning

Spring 2019

Homework 1

Contents

The Chess Game 2
1.1 2
1.2 2
1.3 3
1.4 3
Medical Diagnosis 4
2.1 4
2.2 5
MLE and MAP 5
3.1 5
3.2 6
3.3 6
Sentiment Analysis on Tweets 7
4.1 7
4.2 7
4.3 8
4.4 8
4.5 8
4.6 8
4.7 9
Appendix A 12

Ayhan Okuyan
21601531
CS464-1

The Chess Game

1.1

Let's denote the playstyles bold as B, and timid as T, also winning. Hence, we can create a table with the probabilities of Gates winning the match playing only bold, the outcomes will be the multiplications of individual probabilities since each game is independent from each other in a match. Some cells in the results column is merged since in some matches, the outcome is determined independent of the third game.

Probabilities	**Outcome of the Match**	**Results**
$P(W\|B) * P(W\|B) * P(W\|B)$	W	0.36
$P(W\|B) * P(W\|B) * P(L\|B)$	W	
$P(W\|B) * P(L\|B) * P(W\|B)$	W	0.144
$P(W\|B) * P(L\|B) * P(L\|B)$	L	0.096
$P(L\|B) * P(W\|B) * P(W\|B)$	W	0.144
$P(L\|B) * P(W\|B) * P(L\|B)$	L	0.096
$P(L\|B) * P(L\|B) * P(W\|B)$	L	0.16
$P(L\|B) * P(L\|B) * P(L\|B)$	L	

Table 1 – Probabilities for all outcomes given playing all bold.

The question asks the probability of winning with playing all bold. We can find this by adding all individual probabilities which result in winning. The result becomes 0.648.

1.2

Now we are required to see the chance of winning if the probability if Gates plays all timid. However, there is a small contradiction of thoughts. There are two possibilities given the scenario. On one hand, the question clearly states that if the game is tied at the end of two games, the match goes into a sudden death mode in which Gates is forced to playing bold in the third. If this scenario is accepted, then the probability that Gates wins if he plays timid in three games is zero directly. On the other hand, we can accept the question statement as being able to play all the games timidly which he can in the two games. In order to explain, we can create a table for both scenarios.

Ayhan Okuyan
21601531
CS464-1

First Two Games	Third Game for the 1st Scenario	Outcomes for the 1st Scenario	Third Game for the 2nd Scenario	Outcomes for the 2nd Scenario
LL	D	L	D	L
	L	L	L	L
LD	D	L	D	L
	L	L	L	L
DL	D	L	D	L
	L	L	L	L
DD	NOT A POSSIBILITY	X	W	W
		X	L	L

Table 2 – Probabilities for winning given playing all timid for both scenarios.

Since we can see from the table, for the first scenario, there is no way Gates can win, however in the second game, if we consider that he plays bold for the third game after two straight draws, he can win with probability 0.65*0.65*0.6 = 0.2535.

1.3

Here, we need to find the probability of Gates playing bold if we have a prior knowledge on his loss. In order to find that, we use the Bayes rule given as,

$$P(B|L) = \frac{P(L|B) * P(B)}{P(L)}$$

$$P(L) = P(B)P(L|B) + P(T)P(L|T) = 0.5 * 0.4 + 0.5 * 0.35 = 0.375$$

$$P(L|B) = 0.4$$

$$P(B) = 0.5$$

Hence,

$$P(B|L) = \frac{0.4 * 0.5}{0.375} = 0.5\overline{3}$$

1.4

In this question, we see that the probabilities changing due to Steve predicting Gates' movements. The probability trees for the cases where the Gates plays timid and bold are given below.

Ayhan Okuyan
21601531
CS464-1

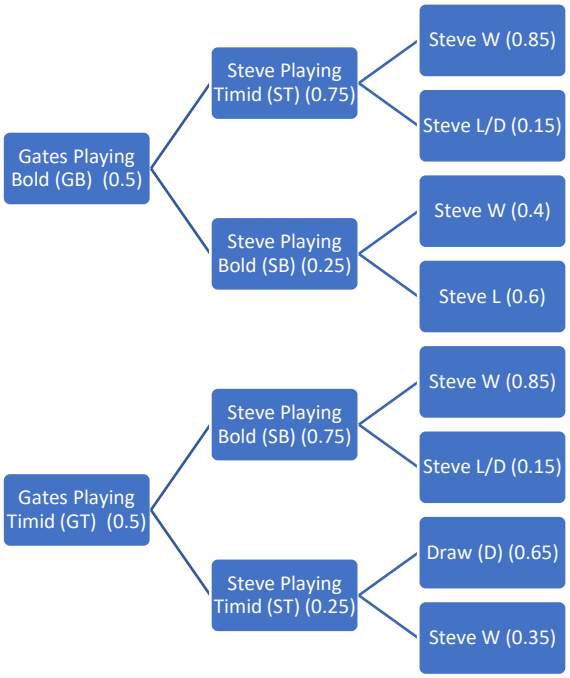

Figure 1 – Probability trees for the given question.

These are the trees that are changed for this question. Here we see that Steve wins in four of the outcomes. We can write the probability of winning as given below. Here we see that the notation has changed since we included which player played which strategy, which their explanations are given on the diagram.

$$P(W) = P(GB)\big(P(ST|GB)P(W|ST,GB) + P(SB|GB)P(W|GB,GB)\big)$$
$$+ P(GT)(P(SB|GT)P(W|SB,GT) + P(ST|GT)P(W|ST,GT))$$
$$P(W) = 0.5 * (0.75 * 0.85 + 0.25 * 0.4) + 0.5 * (0.75 * 0.85 \ + 0.25 * 0.35) = 0.73125$$

Hence, the probability that Steve wins given the circumstances is 0.73125.

Medical Diagnosis

2.1

The probabilities that are requested are given below.

$$P(S = disease) = 0.005$$
$$P(S = healthy) = 0.995$$
$$P(T = positive \mid S = disease) = 0.97$$
$$P(T = negative \mid S = \ disease) = 0.03$$
$$P(T = positive \mid S = healthy) = 0.02$$

Ayhan Okuyan
21601531
CS464-1

$$P(T = negative|\, S = healthy) = 0.98$$

2.2

Then, we use the Bayes rule to find the conditional probability of being ill and not ill with respect to the test being positive.

$$P(S = disease \mid T = positive) = \frac{P(T = positive \mid S = disease)P(S = disease)}{P(T = postive)}$$

$$= \frac{P(T = positive \mid S = disease)P(S = disease)}{P(T = positive \mid S = disease)P(S = disease) + P(T = \ positive \,|S = healthy)P(S = healthy)}$$

$$= \frac{0.97 * 0.005}{0.97 * 0.005 + 0.02 * 0.995} = \frac{0.00485}{0.00485 + 0.0199} = 0.196$$

Furthermore, we observe the compliment,

$$P(S = healthy \mid T = positive) = 1 - P(S = disease \mid T = positive) = 0.804$$

Here, we see that even the test results in a positive manner, the probability the person being ill turns out to be nearly 1/5, which is a very low truth percentage and further testing should be applied for a person with a positive test results to be diagnosed with disease.

MLE and MAP

3.1

For the duration of this question the notation log is used to identify natural logarithm. What needs to be done Is to take the log likelihood function for the ML estimator and find its argmax, hence,

$$L(\,\lambda|x_1, \dots, x_n) = \prod_{i=1}^{n} e^{-\lambda}\frac{\lambda^{x_i}}{x_i!}$$

$$\log\big(L(\,\lambda|x_1, \dots, x_n)\big) = \log\left(\prod_{i=1}^{n} e^{-\lambda}\frac{\lambda^{x_i}}{x_i!}\right) = \sum_{i=1}^{n} \log(e^{-\lambda}\frac{\lambda^{x_i}}{x_i!})$$

$$= \sum_{i=1}^{n} \log\big(e^{-\lambda}\big) + \log(\frac{\lambda^{x_i}}{x_i!}) = \sum_{i=1}^{n} -\lambda + \log(\lambda^{x_i}) - \log(x_i!)$$

$$= -n\lambda + \log(\lambda)\sum_{i=1}^{n} x_i - \sum_{i=1}^{n} \log(x_i!)$$

$$Hence, we\ take\ this\ likelihood\ function\ with\ respect\ to\ \lambda,$$

Ayhan Okuyan
21601531
CS464-1

$$\frac{\partial \log\big(L(\,\lambda|x_1,\dots,x_n)\big)}{\partial\lambda} = -n + \frac{1}{\lambda}\sum_{i=1}^{n} x_i$$

$$Then\ we\ move\ onto\ finding\ the\ argmax\ of\ this\ function, argmax\left(-n + \frac{1}{\lambda}\sum_{i=1}^{n} x_i\right)$$

$$-n + \frac{1}{\hat{\lambda}}\sum_{i=1}^{n} x_i = 0$$

$$\hat{\lambda} = \frac{1}{n}\sum_{i=1}^{n} x_i = sample\ mean$$

3.2

For the MAP estimate, we use the Bayes rule to find the probability of $P(\lambda \mid x_1,\dots,x_n)$, then apply the logarithm, then take the derivative to find the MAP estimation for λ. Hence,

$$\frac{\partial}{\partial\lambda}\left(\log\frac{P(x_1,\dots,x_n|\,\lambda)P(\lambda)}{P(x_1,\dots,x_n)}\right) = \frac{\partial}{\partial\lambda}\left(\log(P(x_1,\dots,x_n|\,\lambda)) + \log\big(P(\lambda)\big) - \log(P(x_1,\dots,x_n))\right)$$

We know from the prior question the first and the third terms since, hence we plug them in and equate the term to zero in order to find the maximum argument (argmax).

$$= -n + \frac{1}{\lambda}\sum_{i=1}^{n} x_i + \frac{\partial}{\partial\lambda}\log\big(P(\lambda)\big) = -n + \frac{1}{\lambda}\sum_{i=1}^{n} x_i + \frac{\partial}{\partial\lambda}\log\left(\frac{\beta^{\alpha}\lambda^{\alpha-1}e^{-\beta x}}{(\alpha-1)!}\right) = 0$$

$$= -n + \frac{1}{\lambda}\sum_{i=1}^{n} x_i + \frac{\partial}{\partial\lambda}(\alpha-1)\log(\lambda) - \frac{\partial}{\partial\lambda}(\beta\lambda) = -n + \frac{1}{\lambda}\sum_{i=1}^{n} x_i + \frac{\alpha-1}{\lambda} - \beta = 0$$

$$\hat{\lambda} = \frac{\sum_{i=1}^{n} x_i + \alpha - 1}{n + \beta}$$

3.3

In this part we are supposed to show if the ML estimate of λ and MAP estimate with uniform prior is the same for any interval. For that we define the uniform distribution of the prior as

$$U(x|a,b) = \frac{1}{b-a}$$

Then, we calculate the MAP estimate with this prior like we have in the previous question using the log likelihood and argmax functions.

$$\frac{\partial}{\partial\lambda}\left(\log\frac{P(x_1,\dots,x_n|\,\lambda)P(\lambda)}{P(x_1,\dots,x_n)}\right)=-n+\frac{1}{\lambda}\sum_{i=1}^{n}x_i+\frac{\partial}{\partial\lambda}\log\big(P(\lambda)\big)$$

$$=-n+\tfrac{1}{\lambda}\textstyle\sum_{i=1}^{n}x_i+\frac{\partial}{\partial\lambda}\log\left(\frac{1}{b-a}\right)=-n+\tfrac{1}{\lambda}\sum_{i=1}^{n}x_i=0 \text{ Hence,}$$

$$\hat{\lambda}=\frac{1}{n}\sum_{i=1}^{n}x_i=sample\ mean\ which\ is\ equal\ to\ the\ ML\ estimate$$

We see this result since the prior distribution of λ is independent from the variable λ itself.

Sentiment Analysis on Tweets

4.1

The Bayes Rule for a Naïve Bayes approach can be considered as the Naïve Bayes rule, but with a prior belief that the prior probabilities for any feature being conditionally independent from each other. Also, we know that the denominator for a Naïve Bayes equation is the prior probability of the features, which can be expressed as the sum of any conditional probability of the feature on the label multiplied by the prior probability of the label. While calculating the probabilities for any label test case, we see that no matter what the label is, the denominator gives the same result. As we are designing the learning algorithm according to which probability would become the highest in a scheme with discrete labels, the division with this denominator would be meaningless and it would only increase the computation time.

4.2

The number of negative tweets in the training dataset is 7091 and there are 11712 total tweets. Hence, the negative tweet ratio is 60.54%. The training set is skewed towards negative tweets and not balanced. Having a skewed dataset should affect the model since we count the number of words, in tweets believing that in prior, the probability for a tweet belonging to any class is 0.6054, which is a biased estimation. At the end, while the models will be better at classifying the negative tweets compared to the positive and neutral ones, which would affect the learning process. For a solution, the data can be trimmed to have mostly the same amounts of kinds of tweets. Perhaps, we can build a generative model and create pseudo training samples to nearly equate the number of each sample that belong to a label.

Ayhan Okuyan
21601531
CS464-1

4.3

We have three classes and 5722 different features. For each feature, we need to calculate three conditional probabilities and apart from that, two prior probability for the labels. We need to know only two since the third probability is the complement of the other two's union. Hence, we need 5722*3+2 parameters, which is equal to 17168.

4.4

My testing accuracy for the ML estimator Naïve Bayes Model is 62.807% and the number for wrong predictions is 1089 out of 2928 tweets. The classifier ended up predicting with an accuracy that is acceptable but cannot be called good. The reason behind this poor accuracy is the existence of zero probabilities. If there are no instances of a word in the training set, the probability for that in the test becomes zero no matter how many times it was mentioned in a given test data. For example, let's say that the word "amazing" does not appear in any of the training data, but in a test data there is three of it. Since it was not mentioned in the training, the multiplication of the number three and the probability zero becomes zero, and since we are using a log likelihood, the positive value for that tweet becomes -inf (negative infinity), automatically making that data unable to be positive. However, we would expect that any tweet with the word "amazing" in it three times would probably be classified as positive. These types of cases lower the accuracy mainly.

4.5

In the MAP estimate case, for each probability of a feature given the label, we add an alpha constant to the numerator and alpha times the number of features to the denominator which is called a Dirichlet prior. In our case, alpha was given as one. The physical meaning is that we accept that there already is one of each word in any given tweet, which is reasonable, since we do not alter the probabilities drastically and eliminate the zero probability cases. When we changed to this estimator, the accuracy improved drastically to 75.30% from 62.8%. Also, the number of mispredictions have decreased from 1089 to 723. To comment, this is a much better statistic as we have correctly classified nearly three out of all four test data.

4.6

The Bernoulli Naïve Bayes Model with ML estimator that has been trained, on the same test data, has predicted with accuracy 64.14% with a number of 1050 mispredictions. This was a number that was expected. This model is nearly on par with the Multinomial Naïve Bayes model

with the ML estimator. Heuristically, the Bernoulli Model uses the knowledge of existence of a word together with the non-existence, but not the occurrences of a word in a tweet. On the other hand, the multinomial MLE model uses the knowledge for the existence of a word and the number of occurrences but not the nonexistence. This kind of give and take has resulted in similar occurrences while the MAP estimator Naïve Bayes model outclassed the two.

4.7

The most commonly used 20 words in positive training tweets with their number of occurrences are given as,

@southwestair	573
@jetblue	545
@united	500
flight	331
@usairways	248
great	**205**
@virginamerica	152
service	133
love	**116**
best	**95**
guys	93
customer	92
time	87
awesome	**86**
help	71
airline	71
amazing	**69**
today	68
fly	62
flying	61

Table 3 – The occurrence numbers of the words in the positive training tweets (20 highest)

By looking at the tweets, we see that the mostly used twenty words are as much expected. The words that are given in bold are the “decisive” words. Other than them, there are the Twitter aliases of the airlines which are as much expected to be on top since we would expect each

tweet to contain one. By looking at the data we see that the “southwest airlines” is a better airline than the rest. However, this statistic can be deceiving since we are not clear on how many tweets are tagged with their name. The figure below illustrates the 20 frequently seen words in the negative samples of the training set.

@united	2671
flight	2270
@usairways	2163
@southwestair	1209
@jetblue	963
cancelled	620
service	571
hours	484
hold	482
time	480
customer	466
help	439
delayed	436
plane	429
hour	378
flights	334
bag	326
gate	321
late	305
flightled	290

Table 4 – The occurrence numbers of the words in the negative training tweets (20 highest)

From this part of the data, we can clearly see that most of the negative tweets are associated with a delay of the flight. The highlighted words show that the cause of negativity being the flight “cancelled”, “delayed”, or “hold”. Also, there is the word “flightled” which is a hybrid of the words, “flight” and “startle”, corresponding to the scare some passengers have observed on the flight, which is again a considerably negative word. Finally, we look at the most common words in the neutral data.

@jetblue	706
@united	701
@southwestair	667
flight	495
@usairways	368
@virginamerica	175
flights	144
help	133
fleek	107
fleet's	102
dm	99
time	83
tomorrow	83
flying	71
cancelled	66
fly	65
change	64
today	61
travel	61
check	60

Table 5 – The occurrence numbers of the words in the neutral training tweets (20 highest)

Here, we observe the most commonly seen neutral words, which are quite as expected. We see words that are mainly used for flying like "fleet", "flight", "flying" which are highlighted with bold typing. Also, we see some amount of positive and negative words. Positive being "help" and "fleek". This can be due to the nature of natural language as we most of the time connect the positive and negative sentences with conjunctions. Also, it can be due to the misclassifications that have happened. May be these sentences should have been classified as positive or negative. However, in the end I believe that the model that I have constructed is highly interpretable since the mostly used words lists contained mostly the desired words that we needed to classify. However when we look at the numbers, we see a huge discrepancy between the occurrences of negative words when compared with positive and neutrals since

the dataset has ~60% negative words in it. The python code that has been written is given in Appendix A.

Appendix A

```
import pandas as pd
import numpy as np

#get the argmax of each prediction list from list of lists and count matches,
then print the accuracy
def calcPerc(test_pred, y_test_dt):
    true_count = 0
    false_count = 0
    count = -1
    for x in test_pred:
        max_index = np.argmax(x)
        count += 1
        if max_index == 0 and y_test_dt.iloc[count].values == 'neutral':
            true_count += 1
        elif max_index == 1 and y_test_dt.iloc[count].values == 'positive':
            true_count += 1
        elif max_index == 2 and y_test_dt.iloc[count].values == 'negative':
            true_count += 1
        else:
            false_count += 1
    print('Number of correct predictions: ' + str(true_count))
    print('Number of false predictions: ' + str(false_count))
    print('Test accuracy: ' + str(100*true_count/(true_count+false_count)) + '%')

# read training csv files and the txt file
nms_frq = pd.read_csv('question-4-vocab.txt', header=None, sep='\t',
names=['names', 'freqs'])
x_train = pd.read_csv('question-4-train-features.csv', header=None, sep=',',
names=nms_frq['names'])
y_train = pd.read_csv('question-4-train-labels.csv', header=None, sep=',',
names=['results'])

# concatenate features and labels for easy grouping
train = pd.concat([x_train, y_train], axis=1)

# set prior probabilities
# get the negative priors for each word
neg_prior = train.loc[train['results'] ==
'negative'].select_dtypes(pd.np.number).sum().rename('total_neg')
# get the positive priors
pos_prior = train.loc[train['results'] ==
'positive'].select_dtypes(pd.np.number).sum().rename('total_pos')
# get the neutral priors
ntr_prior = train.loc[train['results'] ==
'neutral'].select_dtypes(pd.np.number).sum().rename('total_ntr')
```

Ayhan Okuyan
21601531
CS464-1

```
total_prior = x_train.select_dtypes(pd.np.number).sum().rename('total_prior')

# get the total number of words in each res=neg/pos/ntr matrix
total_pos_word_cnt = train.loc[train['results'] ==
'positive'].select_dtypes(pd.np.number).values.sum()
total_neg_word_cnt = train.loc[train['results'] ==
'negative'].select_dtypes(pd.np.number).values.sum()
total_ntr_word_cnt = train.loc[train['results'] ==
'neutral'].select_dtypes(pd.np.number).values.sum()

# get prior probabilities of labels
pr_prob_pos = train.loc[train['results'] == 'positive'].shape[0] /
x_train.shape[0]
pr_prob_neg = train.loc[train['results'] == 'negative'].shape[0] /
x_train.shape[0]
pr_prob_ntr = 1 - (pr_prob_pos + pr_prob_neg)

# get the ln for each label prior (used in a lot of calculations)
pr_prob_pos_ln = np.log(pr_prob_pos)
pr_prob_neg_ln = np.log(pr_prob_neg)
pr_prob_ntr_ln = np.log(pr_prob_ntr)

neg_matrix = train.loc[train['results'] == 'negative']
print('Number of negative tweets in the training data: ' +
str(neg_matrix.shape[0]))
print('Total number of tweets: ' + str(train.shape[0]))
print('Percentage of negative tweets: ' +
str(neg_matrix.shape[0]/train.shape[0]))

# read csv files for test data
x_test = pd.read_csv('question-4-test-features.csv', header=None, sep=',',
names=nms_frq['names'])
y_test = pd.read_csv('question-4-test-labels.csv', header=None, sep=',',
names=['results'])


# ## Multinomial Naive Bayes Model

# ML Estimation

# calculate the probabilities for each word in any label
neg_prior_prb = np.log(neg_prior/total_neg_word_cnt)
pos_prior_prb = np.log(pos_prior/total_pos_word_cnt)
ntr_prior_prb = np.log(ntr_prior/total_ntr_word_cnt)

# run the prediction for each test sample
mle_test_prediction = [[None for y in range(3)] for x in range(y_test.shape[0])]
count_y_ntr = 0
count_y_pos = 1
count_y_neg = 2
```

```
count_x = 0
for i in range(x_test.shape[0]):
    row = x_test.iloc[i]
    temp_pos = pos_prior_prb * row.values
    temp_pos = pr_prob_pos_ln + np.sum(temp_pos.fillna(0))
    temp_neg = neg_prior_prb * row.values
    temp_neg = pr_prob_neg_ln + np.sum(temp_neg.fillna(0))
    temp_ntr = ntr_prior_prb * row.values
    temp_ntr = pr_prob_ntr_ln + np.sum(temp_ntr.fillna(0))
    mle_test_prediction[count_x][count_y_ntr] = temp_ntr
    mle_test_prediction[count_x][count_y_pos] = temp_pos
    mle_test_prediction[count_x][count_y_neg] = temp_neg
    count_x += 1

print("Multinomial Naive Bayes Classifier with ML Estimator\n")
calcPerc(mle_test_prediction, y_test)
print("\n")


# MAP Estimation

# calculate the probabilities for each word in any label given alpha = 1
alpha = 1
neg_prior_prb =
np.log((neg_prior+alpha)/(total_neg_word_cnt+(alpha*x_train.shape[1])))
pos_prior_prb =
np.log((pos_prior+alpha)/(total_pos_word_cnt+(alpha*x_train.shape[1])))
ntr_prior_prb =
np.log((ntr_prior+alpha)/(total_ntr_word_cnt+(alpha*x_train.shape[1])))

# run the prediction for each test sample
map_test_prediction = [[None for y in range(3)] for x in range(y_test.shape[0])]
count_y_ntr = 0
count_y_pos = 1
count_y_neg = 2
count_x = 0

for i in range(x_test.shape[0]):
    row = x_test.iloc[i]
    temp_pos = pos_prior_prb * row.values
    temp_pos = pr_prob_pos_ln + np.sum(temp_pos.fillna(0))
    temp_neg = neg_prior_prb * row.values
    temp_neg = pr_prob_neg_ln + np.sum(temp_neg.fillna(0))
    temp_ntr = ntr_prior_prb * row.values
    temp_ntr = pr_prob_ntr_ln + np.sum(temp_ntr.fillna(0))
    map_test_prediction[count_x][count_y_ntr] = temp_ntr
    map_test_prediction[count_x][count_y_pos] = temp_pos
    map_test_prediction[count_x][count_y_neg] = temp_neg
    count_x += 1

print("Multinomial Naive Bayes Classifier with MAP Estimator\n")
```

```
calcPerc(map_test_prediction,y_test)
print("\n")

# ## Bernoulli Naive Bayes Model

# convert training data to 1/0s for the bernoulli classifier
x_train_br = x_train.copy()
x_train_br[x_train_br != 0] = 1

# concatenate features and labels for easy grouping
train_br = pd.concat([x_train_br, y_train], axis=1)

# set prior probabilities
# get the negative priors for each word
neg_prior_br = train_br.loc[train['results'] ==
'negative'].select_dtypes(pd.np.number).sum().rename('total_neg')
# get the positive priors
pos_prior_br = train_br.loc[train['results'] ==
'positive'].select_dtypes(pd.np.number).sum().rename('total_pos')
# get the neutral priors
ntr_prior_br = train_br.loc[train['results'] ==
'neutral'].select_dtypes(pd.np.number).sum().rename('total_ntr')

total_prior_br =
x_train_br.select_dtypes(pd.np.number).sum().rename('total_prior')

# total number of pos/neg/ntr tweets
total_pos_tweets = train_br.loc[train['results'] == 'positive'].shape[0]
total_neg_tweets = train_br.loc[train['results'] == 'negative'].shape[0]
total_ntr_tweets = train_br.loc[train['results'] == 'neutral'].shape[0]

# get prior probabilities of labels
pr_prob_pos_br = train_br.loc[train['results'] == 'positive'].shape[0] /
x_train_br.shape[0]
pr_prob_neg_br = train_br.loc[train['results'] == 'negative'].shape[0] /
x_train_br.shape[0]
pr_prob_ntr_br = 1 - (pr_prob_pos + pr_prob_neg)

# get the ln for each label prior (used in a lot of calculations)
pr_prob_pos_ln = np.log(pr_prob_pos_br)
pr_prob_neg_ln = np.log(pr_prob_neg_br)
pr_prob_ntr_ln = np.log(pr_prob_ntr_br)


# transform the test features to 1/0s
x_test_br = x_test.copy()
x_test_br[x_test_br != 0] = 1

#calcluate the probabilities for each word and also their compliments
neg_prior_prb = neg_prior_br/total_neg_tweets
pos_prior_prb = pos_prior_br/total_pos_tweets
```

Ayhan Okuyan
21601531
CS464-1

```
ntr_prior_prb = ntr_prior_br/total_ntr_tweets
neg_prior_prb_cmp = 1-neg_prior_prb
pos_prior_prb_cmp = 1-pos_prior_prb
ntr_prior_prb_cmp = 1-ntr_prior_prb

#run the prediction for each test sample
mle_test_prediction_br = [[None for y in range(3)] for x in
range(y_test.shape[0])]
count_y_ntr = 0
count_y_pos = 1
count_y_neg = 2
count_x = 0
for i in range(x_test_br.shape[0]):
    row = x_test_br.iloc[i]
    row_cmp = 1 - row
    temp_pos = pos_prior_prb * row.values + pos_prior_prb_cmp * row_cmp.values
    temp_pos = pr_prob_pos_ln + np.sum(np.log(temp_pos.fillna(0)))
    temp_neg = neg_prior_prb * row.values + neg_prior_prb_cmp * row_cmp.values
    temp_neg = pr_prob_neg_ln + np.sum(np.log(temp_neg.fillna(0)))
    temp_ntr = ntr_prior_prb * row.values + ntr_prior_prb_cmp * row_cmp.values
    temp_ntr = pr_prob_ntr_ln + np.sum(np.log(temp_ntr.fillna(0)))
    mle_test_prediction_br[count_x][count_y_ntr] = temp_ntr
    mle_test_prediction_br[count_x][count_y_pos] = temp_pos
    mle_test_prediction_br[count_x][count_y_neg] = temp_neg
    count_x += 1

print("Bernoulli Naive Bayes Classifier with MAP Estimator\n")
calcPerc(mle_test_prediction_br,y_test)
print("\n")


# ## Finding the Most Commonly Used Words

print('Most commonly used words in positive tweets:')
print(pos_prior.nlargest(n=20,keep='all'))
print('Most commonly used words in negative tweets:')
print(neg_prior.nlargest(n=20,keep='all'))
print('Most commonly used words in neutral tweets:')
print(ntr_prior.nlargest(n=20,keep='all'))
```