Hatice Kübra Çağlar
21803456

CS 464 - Homework 1

Question 1

The given distribution of the grade types are as follows:

$$P(H) = \frac{64}{100}, \text{ the probability of a grade being high.}$$

$$P(L) = \frac{24}{100}, \text{ the probability of a grade being low.}$$

$$P(F) = \frac{12}{100}, \text{ the probability of a grade being failing.}$$

Also, conditional probabilities of high, low, and failing grades are received by motivated students are as follows (given):

$$P(S_M|H) = \frac{87}{100}, \text{ the probability of a high grade is received by a motivated student.}$$

$$P(S_M|L) = \frac{21}{100}, \text{ the probability of a low grade is received by a motivated student.}$$

$$P(S_M|F) = \frac{4}{100}, \text{ the probability of a failing grade is received by a motivated student.}$$

Question 1.1

$$\begin{aligned} P(S_M) &= P(S_M, H) + P(S_M, L) + P(S_M, F) \\ &= P(S_M|H)P(H) + P(S_M|L)P(L) + P(S_M|F)P(F) \\ &= 0.612 \end{aligned}$$

Question 1.2

Using Bayes' Theorem:

$$P(H|S_M) = \frac{P(S_M|H)P(H)}{P(S_M)} = 0.9098$$

Question 1.3

First of all, we need to be aware that:

$$P(S_U|H) = 1 - P(S_M|H) = \frac{13}{100}$$

$$P(S_U) = 1 - P(S_M) = \frac{388}{1000}$$

After calculating these values:

$$P(H|S_U) = \frac{P(S_U|H)P(H)}{P(S_U)} = 0.2144$$

Hatice Kübra Çağlar
21803456

Question 2

Question 2.1

1) Except for the football class, the class distributions are close to each other. There are 77 athletics, 86 cricket, 198 football, 114 rugby, and 77 tennis instances.

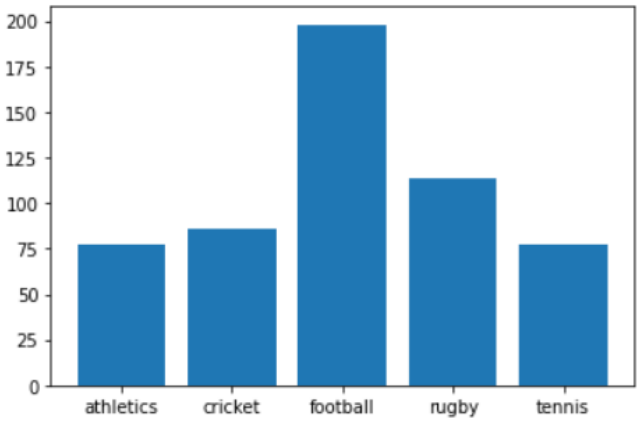

Figure: Plot of the class distribution of the training set

2) The training set is skewed toward the football class. The effect of the Naive Bayes model is, the class having considerably more instances than others has a higher prior probability, and it can make the model more biased towards that class. Since that class has a higher prior probability, an instance is more likely to be classified in that class rather than others. A possible solution is undersampling in which we randomly select a reasonable amount of the data instances from that class and get rid of the surplus.

3) The distribution of data samples is similar for validation and training sets. They both have surplus value in football class. In the validation set, there are 24 athletics, 38 cricket, 67 football, 33 rugby, and 23 tennis instances as shown in the figure below. If a training set and a validation set have different data distributions, the prior probability term in the Naive Bayes algorithm would be misleading.

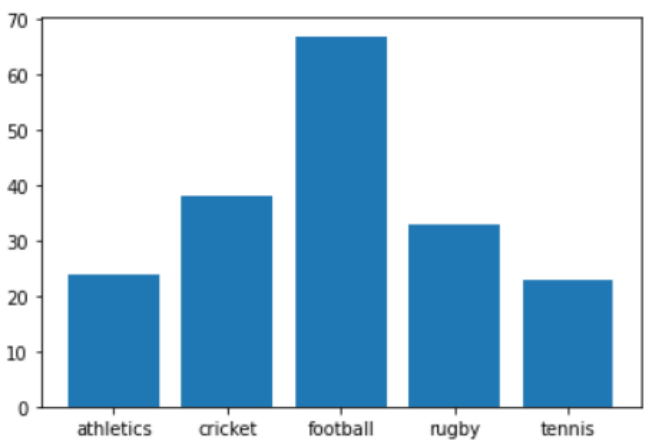

Figure: Plot of the class distribution of validation set

4) If the difference between the number of data instances for each class is so obvious, the accuracy is affected due to the manipulation of prior probability. However, if the distribution is not too unbalanced, then with the effect of likelihoods, the accuracy is not affected to a vast extent.

Question 2.2

To calculate likelihoods for the Naive Bayes algorithm easily, I separate all the data instances in the training set according to their class labels. Then adding up the frequencies of the words belonging to the same class, I created a separate table for the calculations. The rows of the table are displayed below.

	claxton	hunt	first	major	medal	british	hurdler	sarah	confid	win	...	mcenro	mauresmo	ameli	hip	fiveset	mario	ancic	lundgren	class_label	summation
0	0	0	124	13	1	8	0	0	17	146	...	0	0	0	0	0	0	0	0	3	20755
0	0	4	141	4	0	2	0	0	17	43	...	0	0	0	0	0	0	0	0	1	17179
0	0	2	122	27	7	0	0	0	34	173	...	0	0	0	0	0	0	0	0	2	31533
0	0	0	102	7	0	20	0	0	22	130	...	5	10	3	6	0	3	3	3	4	10819
0	10	1	37	17	38	44	5	5	18	73	...	0	0	0	0	0	0	0	0	0	11572

The only thing that may confuse a reader of this report is that the order of the classes, they are not sorted. The order is [3, 1, 2, 4, 0] as shown above and below.

```
#the order of the classes in my algorithm
indexes
```

```
[3, 1, 2, 4, 0]
```

Figure: The order of classes

The reason is, the classes appear in the training set in this order, and I thought that it would not change anything, so left it like that. The only difference it makes is, as the homework document states when we compare two probability values and see they are equal, they should be classified as the class with the lower index. If I had ordered the class list in ascending order, class 0 have the advantage over class 1 in case of ties, however, this is not the case right now. Actually, this is not a relevant discussion about the algorithm of the model, I just want to state it to avoid further confusion.

Hatice Kübra Çağlar
21803456

The representation of the confusion matrix is shown below.

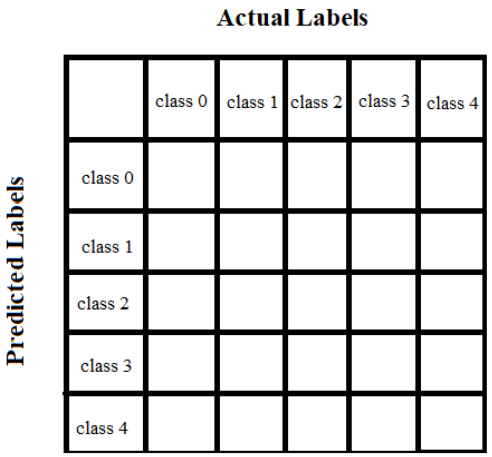

Figure: Representation of the confusion matrix

For the value of log0, I used *-np.inf*. Since there are words with zero frequency in each row, the -∞ value is used in the calculation of each posterior probability. When Python compares two infinity values, it counts them as equal, hence this is the aforementioned tie situation. Since all the posterior probabilities are -∞ in the first case without a fair Dirichlet prior, all data points are assigned to the class with the lowest index, which is class 3 in my case. As shown below, all data instances are assigned to class 3 in the first model.

```
Confusion matrix when alpha = 0 is:
[[ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [24. 38. 67. 33. 23.]
 [ 0.  0.  0.  0.  0.]]
```

Figure: Confusion matrix of the first algorithm

Accuracy of the first algorithm is 17.84% as shown below.

```
Accuracy of the first Multinomial Naive Bayes model (alpha = 0) is 17.83783783783784 percent
```

Figure: Accuracy of the first model

Number of the wrong label predictions made in the first algorithm is 152 as stated below.

```
Number of wrong predictions made in the first model is 152.0
```

Figure: Number of wrong predictions for the first algorithm

Hatice Kübra Çağlar
21803456

Question 2.3

For this version of the algorithm, the only difference is the fair Dirichlet prior which is $\alpha = 1$. This coefficient enables us to avoid the log0 situation hence, the algorithm gives a reasonable confusion matrix and a good accuracy score.

```
Confusion matrix when alpha = 1 is:
[[24.  0.  0.  0.  1.]
 [ 0. 35.  0.  0.  0.]
 [ 0.  1. 66.  0.  0.]
 [ 0.  2.  1. 33.  0.]
 [ 0.  0.  0.  0. 22.]]
```

Figure: Confusion matrix for the second algorithm

The accuracy score of the Naive Bayes model with fair Dirichlet prior is 97.297% as shown below.

```
Accuracy of the second Multinomial Naive Bayes model (alpha = 1) is 97.2972972972973 percent
```

Figure: Accuracy score of the second model

Lastly, there are only 5 wrong label predictions in this model.

```
Number of wrong predictions made in the second model is 5.0
```

Figure: Number of wrong predictions for the second algorithm

Question 3.4

The Dirichlet prior α affects my model in a very useful way. Without it, there are -∞ terms due to the words having zero frequency in a document. As we add α to our denominator and numerator, it becomes impossible to have a log0, namely, -∞ term in our calculations. This enables the model to compare the probability values and assign the most likely labels. This is the reason why the accuracy increased drastically.

Hatice Kübra Çağlar
21803456

CODE

Only functions will be copied here, the whole code can be found in the .zip file.

```
def SeperateClasses(train):

  yk = [i for i in train['class_label'].unique()]
  #total number of classes
  N = train['class_label'].size
  prior_list = []
  class_arr = []

  for i in yk:
    train_0 = train[train['class_label'] == i]
    Nk = train_0['class_label'].size
    prior_list.append(Nk/N)
    train_0 = train_0.drop(labels = ['class_label'], axis=1)
    train_0 = train_0.sum(axis=0)
    sum = np.sum(train_0)
    train_0 = pd.DataFrame(train_0)
    train_0 = train_0.T
    train_0['class_label'] = i
    train_0['summation'] = sum
    class_arr.append(train_0)

  return class_arr, prior_list, yk

def MNB(class_list, prior_list, val_data, indexes, alpha):

  best_prob = -np.inf
  keep_index = 0
  predicted_labels = []
  col_size = val_data[0:1].size - 1 #4613
  row_size = val_data['class_label'].size

  for i in range(row_size):
    best_prob = -np.inf
    for k in range(len(prior_list)):
      prob = prior_list[k]
      log_prob = np.log(prob)
      for j in range(col_size):
        multiplier = val_data.iloc[i,j]
```

```
                                                    likelihood      =
(class_list[k].iloc[0,j]+alpha)/(class_list[k].iloc[0,-1]           +
alpha*col_size)
        if likelihood == 0:
          log_likelihood = -np.inf
        else:
          log_likelihood = np.log(likelihood)
        log_prob += multiplier*log_likelihood
      if log_prob > best_prob:
        best_prob = log_prob
        keep_index = k
    predicted_labels.append(indexes[keep_index])
  return predicted_labels

def metrics(predicted_labels, dataset):
  labels = dataset['class_label']
  confusion_matrix = np.zeros((5,5))
  accuracy = 0
  true_pred = 0
  wrong_pred = 0
  for i in range(len(predicted_labels)):
    #c_no represents the actual label
    c_no = labels[i]
    #p_no represents the predicted labels
    p_no = predicted_labels[i]
    confusion_matrix[p_no,c_no] += 1
    if c_no == p_no:
      true_pred += 1
  total_pred = np.sum(confusion_matrix)
  accuracy = true_pred/total_pred
  wrong_pred = total_pred - true_pred
  return confusion_matrix, accuracy, wrong_pred
```