COMP8043 - MACHINE LEARNING

ASSIGNMENT 1 - Bayesian Classification

Neil O'Sullivan

R00206266

Task 1:

```
def task1():
    data["Split"] = data["Split"].map(lambda x: 1 if x == "test" else 0) #Map split 1=Test, 0 = Training

    training_data_reviews = data[data["Split"] == 0][["Review"]]
    training_labels = data[data["Split"] == 0][["Sentiment"]]
    test_data_reviews = data[data["Split"] == 1][["Review"]]
    test_labels = data[data["Split"] == 1][["Sentiment"]]

    test_data = data[data["Split"] == 1].copy(deep=True)#Deep copies to slice again
    training_data = data[data["Split"] == 0].copy(deep=True)

    training_data["Sentiment"] = training_data["Sentiment"].map(lambda x: 1 if x == "positive" else 0) #map positive or negative to 1 or 0

    number_positive_training = training_data[training_data["Sentiment"] == 1][["Sentiment"]].count() # count each in the training set where sentiment =
    number_negative_training = training_data[training_data["Sentiment"] == 0][["Sentiment"]].count() # count each in the training set where sentiment =

    test_data["Sentiment"] = test_data["Sentiment"].map(lambda x: 1 if x == "positive" else 0) #map positive or negative to 1 or 0
    number_positive_test = test_data[test_data["Sentiment"] == 1][["Sentiment"]].count() # count each in the test set where sentiment = 1 (positive)
    number_negative_test = test_data[test_data["Sentiment"] == 0][["Sentiment"]].count() # count each in the test set where sentiment = 0 (negative)

    #Outputs as per brief
    print("Number of positive reviews in the training set: ",number_positive_training)
    print("Number of negative reviews in the training set: ",number_negative_training)
    print("Number of positive reviews in the test set: ",number_positive_test)
    print("Number of negative reviews in the test set: ",number_negative_test)



    return training_data_reviews, training_labels, test_data_reviews, test_labels # returns 4 lists as per brief
```

As can be seen above the code for task one where it reads the file and separates the data as per the brief, returning the four lists and outputting the number of positive and negative reviews in the training and test sets.

```
Number of positive reviews in the training set:  Sentiment    12500
dtype: int64
Number of negative reviews in the training set:  Sentiment    12500
dtype: int64
Number of positive reviews in the test set:  Sentiment    12499
dtype: int64
Number of negative reviews in the test set:  Sentiment    12500
dtype: int64
```

Above is the output for task 1.

Task 2:

```
def task2(training_data_reviews, minWordLength, minWordOccurence):

    training_data_reviews = training_data_reviews['Review'].str.replace('[^a-zA-Z0-9]', ' ', regex=True).str.strip()        #Remove special characters
    training_data_reviews = training_data_reviews.str.lower()           #Change to lower case
    all_words = training_data_reviews.str.split()                       #Split into individual words

    new_words = []    #array for every word
    for i in all_words:  #for every word in all_words add to the new array
        new_words += i

    wordOccurences = {}       #dictionary for word occurrences
    words_result = []         #array for final result to be returned

    for word in new_words:                                              #Count occurrences of each word
        if (len(word) >= minWordLength):                                #if the length is equal to or above the minimum length input
            if (word in wordOccurences):
                wordOccurences[word] = wordOccurences[word] + 1         #if already in occurrences add to count
            else:
                wordOccurences[word] = 1                                #if not already in occurrences add word occurrences

    for word in wordOccurences:                                         #for every word in the dictionary
        if wordOccurences[word] >= minWordOccurence:                    #if it has occurred more than the minimum occurrence input
            words_result.append(word)                                   #add to the result

    return words_result                                                  #return list of words as list as per brief
```

```
def main():
    training_data_reviews, training_labels, test_data_reviews, test_labels = task1()  #task1

    minWordLength = int(input("Enter minimum word length:"))
    minWordOccurence = int(input("Enter minimum word occurrence:"))

    task3words = task2(training_data_reviews, minWordLength, minWordOccurence)  #task2 returning task3 input
```

Above is the code for task 2 which goes through the reviews in the training subset from task 1, removing non-alphanumeric characters, changing to lowercase and splitting each review into individual words with the minimum word length and occurrence input by the user returning a list of words that meet the criteria.

Task 3:

```python
def task3(task3words, data_reviews, labels):
    wordOccurencesPositive = {}     #dictionary for word occurrences in positive reviews
    wordOccurencesNegative = {}     #dictionary for word occurrences in negative reviews

    for word in task3words: #add words to the dictionaries
        wordOccurencesPositive[word] = 0
        wordOccurencesNegative[word] = 0

    for i, review in enumerate(data_reviews["Review"]):  #loop through each review with enumeration
        words = review.split()                           #split the review into seperate words
        if labels["Sentiment"].values[i] == "positive":  #if the corresponding label is positive
            for word in task3words:                      #for every word in the input set, if the word is in the review add to the word occurrence in po
                if word in words:
                    wordOccurencesPositive[word] += 1
        else:                                           #if not positive
            for word in task3words:                     #for every word in the input set, if the word is in the review add to the word occurrence in neg
                if word in words:
                    wordOccurencesNegative[word] += 1

        for word in task3words:                         #if not in the review add 0
            if word not in wordOccurencesPositive:
                wordOccurencesPositive[word] = 0

        for word in task3words:                         #if not in the review add 0
            if word not in wordOccurencesNegative:
                wordOccurencesNegative[word] = 0


    print("Words occurring in positive reviews:",wordOccurencesPositive)    #output as per brief
    print("Words occurring in negative reviews:",wordOccurencesNegative)  #output as per brief

    return wordOccurencesPositive, wordOccurencesNegative      #retrun dictionaries
```

```python
task3words = task2(training_data_reviews, minWordLength, minWordOccurence)  #task2 returning task3 input
wordOccurencesPositive, wordOccurencesNegative = task3(task3words, training_data_reviews, training_labels) #task 3 returning task4 input
```

```
Enter minimum word length:6
Enter minimum word occurrence:1000
Words occurring in positive reviews: {'pretty': 1154, 'original': 862, 'comedy': 843, 'anyone': 847, 'looking': 793, 'entertaining': 548, 'should': 1665, 'living': 429, 'actors': 1296, 'little': 2314, 'played':
Words occurring in negative reviews: {'pretty': 1585, 'original': 1068, 'comedy': 644, 'anyone': 953, 'looking': 1024, 'entertaining': 347, 'should': 2224, 'living': 295, 'actors': 1473, 'little': 2193, 'played'
```

Above shows task 3 with the list returned from task 2 as an input parameter. I also used the minimum word length of 6 and minimum occurrence of 1000 (to speed up the process) and as can be seen above the function iterates through the positive and negative reviews and if a word is present it is added to the occurrence value in the dictionary and outputs the corresponding dictionaries at the end and returns both dictionaries.

Task 4:

```
def task4(wordOccurencesPositive, wordOccurencesNegative, training_labels):
    total = len(training_labels)  #total number of reviews
    positive = sum(training_labels.iloc[:, 0] == "positive")  #number of positive reviews
    negative = sum(training_labels.iloc[:, 0] == "negative")  #number of negative reviews


    prior_pos = positive / total  #positive prior
    prior_neg = negative / total  #negative prior
    alpha = 1  #defining alpha as 1

    likelihood_positive = {}  #dictionary for P[word in review |positive review]
    likelihood_negative = {}  #dictionary for P[word in review | negative review]

    for word in wordOccurencesPositive: #loop through word occurrences positive dict and apply laplace smoothing with smoothing factor 1 to get probabi
        likelihood_positive[word] = (wordOccurencesPositive[word] + alpha) / (
                    positive + alpha * len(wordOccurencesPositive))

    for word in wordOccurencesNegative: #loop through word occurrences negative dict and apply laplace smoothing with smoothing factor 1 to get probabi
        likelihood_negative[word] = (wordOccurencesNegative[word] + alpha) / (
                    negative + alpha * len(wordOccurencesNegative))

    #print(likelihood_negative)
    #print(likelihood_positive)
    return likelihood_negative, likelihood_positive, prior_pos, prior_neg #return dictionaries and priors as per brief
```

```
likelihood_negative, likelihood_positive, prior_pos, prior_neg= task4(wordOccurencesPositive, wordOccurencesNegative, training_labels)  #task4 ret
```

As can be seen above the code for task 4 calculates the likelihoods and priors as per the brief using the dictionaries returned in task 3 along with the training labels used to get the total number of reviews and number of positive and negative reviews. There is no output for task 4.

Task 5:

```
def task5(review, likelihood_negative, likelihood_positive, prior_pos, prior_neg):
    prediction = [] #array for prediction
    words = review.split() #split review in to single words
    logLikelihood_positive = 0
    logLikelihood_negative = 0

    for word in words: # loop through every word in words
        for key, value in likelihood_positive.items(): # for each key,value pair in the positive dict
            if word == key:
                logLikelihood_positive = logLikelihood_positive + math.log(value) #add the math.log of the value to the corresponding key to the log lik

        for key, value in likelihood_negative.items(): # for each key,value pair in the negative dict
            if word == key:
                logLikelihood_negative = logLikelihood_negative + math.log(value)  #add the math.log of the value to the corresponding key to the log li


    if logLikelihood_positive - logLikelihood_negative > math.log(prior_neg) - math.log(prior_pos):  # if the log likeilhood P minues N is greater than
        prediction.append(1) # add 1 to the prediction array
        print("Positive")  #output as per brief

    else:
        prediction.append(0) # add 0 to the prediction array
        print("Negative")   #output as per brief

    return prediction
```

```
review= input("Enter new review:")

task5(review, likelihood_negative, likelihood_positive, prior_pos, prior_neg) #task5 with new entered review
```

```
Enter new review:This movie was an entertaining comedy where the actors played each part perfec
Positive
```

The images above show in the input and output for task 5 which is a Naïve Bayes classifier tha takes the new review and likelihoods calculated in task 4 to predict whether the review is positive or negative. In the case above it predicts the review as positive which it was.

Task 6:

A

```
k= int(input("Enter how many folds (k):"))  #Input for number of folds

task6a(training_data_reviews, training_labels, minWordLength, minWordOccurence, k)  #task6
```

```
Enter how many folds (k):5
```

```
def task6a(new_data_reviews, new_data_labels, wordlength, wordOcc, k):
    skf = StratifiedKFold(n_splits=k) #kfold cross validator
    accuracies = [] #array for accuracies

    for train_index, test_index in skf.split(new_data_reviews, new_data_labels): #get the tarin and test index from the split using the reviews and la
        new_pred = [] #array for new predicition
        X_train, y_train = new_data_reviews.iloc[train_index, :], new_data_labels.iloc[train_index] #Get the training subset amd corrseponding labels
        X_test, y_test = new_data_reviews.iloc[test_index, :], new_data_labels.iloc[test_index]["Sentiment"].map(
            lambda x: 1 if x == "positive" else 0) #Get the test subset and corrseponding labels mapping to 1 or 0

        word_counts = task2(X_train, wordlength, wordOcc) #get word count from task2
        wordOccPos, wordOccNeg = task3(word_counts, X_train, y_train) # get word occurences from task3

        likeNeg, likePos, priorPos, priorNeg = task4(wordOccPos, wordOccNeg, y_train) #get likelihoods and priors from task 4

        for i in range(len(X_test.index)):  #loop through the length of the test subset from the split
            # print(i)
            review = X_test.iloc[i, :]["Review"]  #get the review from the subest
            pred = task5(review, likeNeg, likePos, priorPos, priorNeg)  #using the review get a prediction from task 5
            new_pred.append(pred[0])  #add the prediction to the new prediciton array

        accuracy = metrics.accuracy_score(y_test, new_pred)  # get accuracy comparing the new prediciton and actual sentiment
        print("Accuracy:", accuracy)
        accuracies.append(accuracy)   #add to accuracies

    acc_mean = sum(accuracies) / len(accuracies)

    print("Acuuracy mean:", acc_mean)
    return acc_mean
```

For task 6 a which is running the k -fold cross validator on the training reviews with k input by the user, in this case 5 and uses the training data reviews and labels from task 1 and the word length and occurrence from task 2 to train the classifier and evaluate the classification accuracy along with returning the mean accuracy score.

```
Positive
Negative
Negative
Negative
Positive
Negative
Negative
Negative
Positive
Negative
Negative
Negative
Negative
Positive
Positive
Positive
Negative
Positive
Negative
Negative
Negative
Accuracy: 0.6884
Acuuracy mean: 0.67672
```

Above is the output for the task6a function and it shows it outputting its prediction for each review with the accuracy for each fold and the last output the accuracy mean of all accuracies in all folds. In the case above the mean is .067672 with word length 6 and occurrence 1000.

B

```
def task6b(new_data_reviews, new_data_labels, k):

    wordOcc = int(input("Task 6b : Enter minimum word occurrence:"))

    word_lengths = [1,2,3,4,5,6,7,8,9,10]
    best_accuracy = 0
    best_length = 0

    for i in word_lengths:
        result = task6a(new_data_reviews, new_data_labels, i, wordOcc, k)
        if result > best_accuracy:
            best_accuracy = result
            best_length = i

    print("Optimal length: ", best_length)
    print("Accuracy: ", best_accuracy)
    return best_length
```

```
best_length=task6b(training_data_reviews, training_labels, k)  # Optimal length from task 6b
```

```
Accuracy: 0.6884
Acuuracy mean: 0.67672
Task 6b : Enter minimum word occurrence:1000
```

The function task6b is used to determine the optimal word length and gives the opportunity to change to word occurrence. It checks each word length and stores the mean accuracy from task 6a and returns the optimal word length by checking each word length. (It takes quite a while to run so can change the occurrence if necessary).

```
Negative
Accuracy: 0.5666
Acuuracy mean: 0.56172
Optimal length:  3
Accuracy:  0.748
```

Above is the end of the output for task6b with the last review predicted negative and the accuracy at .5666 when the word length was 10. Not shown in the image was the prediction for every other review (thousands) at each word length and the optimal length can be seen as 3 with an accuracy mean of .748.

C

```
def taskóc(training_reviews, training_labels, testing_reviews, testing_labels, best_length):
    wordOcc = int(input("Task 6c (Final Evaluation) Enter minimum word occurrence:"))
    new_pred = []
    final_eval = []
    word_counts = task2(training_reviews, best_length, wordOcc)
    wordOccPos, wordOccNeg = task3(word_counts, training_reviews, training_labels)

    likeNeg, likePos, priorPos, priorNeg = task4(wordOccPos, wordOccNeg, training_labels)

    for i in range(len(testing_reviews.index)):
        review = testing_reviews.iloc[i, :]["Review"]
        pred = task5(review, likeNeg, likePos, priorPos, priorNeg)
        new_pred.append(pred[0])

    y_test = testing_labels["Sentiment"].map(lambda x: 1 if x == "positive" else 0)
    confusion = metrics.confusion_matrix(y_test, new_pred)  # create confusion matrix
    print("Confusion matrix:\n", confusion)
    true_positive = []
    true_negative = []
    false_positive = []
    false_negative = []

    true_negative.append(confusion[0, 0])  # append index 0,0 to true negative
    true_positive.append(confusion[1, 1])  # append index 1,1 to true positive
    false_negative.append(confusion[1, 0])  # append index 1,0 to false negative
    false_positive.append(confusion[0, 1])  # append index 0,1 to false positive
```

```
    print("True positive:", true_positive)  # print the number of true positives
    print("True negative:", true_negative)  # print the number of true negatives
    print("False positive:", false_positive)  # print the number of false positives
    print("False negative:", false_negative)  # print the number of false negatives
    print()

    perc_true_positive = sum(true_positive) / (sum(true_positive) + sum(false_positive))   #% of true positives
    perc_false_positive = 1 - perc_true_positive                                           #% of false positives
    perc_true_negative = sum(true_negative) / (sum(true_negative) + sum(false_negative))   #% of true negatives
    perc_false_negative = 1 - perc_true_negative                                           #% of false negatives

    class_acc_score = sum(true_positive) + sum(true_negative) / (
                sum(true_positive)+ sum(true_negative) + sum(false_positive) + sum(false_negative)) #Classification accuracy score
    #print(class_acc_score)
    final_eval.append(confusion)
    final_eval.append("True positive % : ")
    final_eval.append(perc_true_positive)
    final_eval.append("False positive % : ")
    final_eval.append(perc_false_positive)
    final_eval.append(" True negative % : ")
    final_eval.append(perc_true_negative)
    final_eval.append("False negative % : ")
    final_eval.append(perc_false_negative)
    final_eval.append("Classification accuracy score % : ")
    final_eval.append(class_acc_score)
    final_eval.append("")

    print(final_eval)
    return final_eval
```

```
task6c(training_data_reviews, training_labels, test_data_reviews, test_labels, best_length)
```

```
Optimal length:  3
Accuracy:  0.748
Task 6c (Final Evaluation) Enter minimum word occurrence:1000
```

Above shows the final part of task 6 which is the final evaluation using the optimal word length (3) and using the test set from task 1 to evaluate with the classifier.

```
Confusion matrix:
 [[11417  1083]
 [ 4938  7561]]
True positive: [7561]
True negative: [11417]
False positive: [1083]
False negative: [4938]

[array([[11417,  1083],
       [ 4938,  7561]], dtype=int64), 'True positive % : ', 0.8747107820453494, 'False positive % : ', 0.12528921795465064, ' T
```

```
'False positive % : ', 0.12528921795465064, ' True negative % : ', 0.698073983491287, 'False negative % : ', 0.301926016508713, 'Classification accuracy score % : ', 0.7591503660146406, '
```

The image above shows the final evaluation output includng the confusion matrix, the percentage of true positives, true negatives, false positives and false negatives along with the classifaction accuracy score.