Documentation of code developed

Student: Mantvydas Luksas

Student ID: R00150390

Task 1

Task 1 code primarily, takes all the reviews from a file into a dataframe using pandas. For machine learning, its important that all text is converted into a digit format and therefore, the conversion of "train", "test", "negative" and "positive" takes place. Lines of code below from 27 to 31.

```
def Task1(digit):

    df = pd.read_excel("movie_reviews.xlsx")

    df["Split"] = df["Split"].map({"train": 0, "test": 1})

    df["Sentiment"] = df["Sentiment"].map({"negative": 0, "positive": 1})
```

The data is split for training and evaluation by selecting the Split value required. The same is also done to select the type of reviews that are looked for. Code lines below from 33 to 39.

```
trainData = df["Split"] == 0

evaluationData = df["Split"] == 1
```

```
 positiveReviews = df["Sentiment"] == 1

 negativeReviews = df["Sentiment"] == 0
```

This allows to select training Data for both the positive reviews and negative reviews by combining the two query conditions into one query. Code lines below from 41 to 43.

```
train_positive_reviews = np.array(df["Review"][trainData & positiveReviews])

train_negative_reviews = np.array(df["Review"][trainData & negativeReviews])
```

However, we also need to have the whole list of the training data and its classes. The same applies to the evaluation data. Code lines below from 45 to 51.

```
  trainingData = np.array(df["Review"][trainData])

  trainingLabels = np.array(df["Sentiment"][trainData])

  testData = np.array(df["Review"][evaluationData])

  testLabels = np.array(df["Sentiment"][evaluationData])
```

In addition, we need to get the total count of positive and negative reviews for both the training and evaluation data. This is done by using the combined query of what we are looking for with the function count(), so that we can get the total number of that particular data. Code lines below from 53 to 59.

```
trainPositive = df["Review"][trainData & positiveReviews].count()

trainNegative = df["Review"][trainData & negativeReviews].count()

testPositive = df["Review"][evaluationData & positiveReviews].count()

testNegative = df["Review"][evaluationData & negativeReviews].count()
```

To return the different type of lists, a certain digit needs to be passed in to the Task1 to return a certain list. For example, digit 1 returns the training data list. It also prints the total number of different types of reviews. If we pass the integer value of 2 into the method, the labels list for the training data is returned. Code lines below from 61 to 75.

```
if digit == 1:

    print("The number of positive reviews in the training set:", trainPositive)

    print("The number of negative reviews in the training set:", trainNegative)

    print("The number of positive reviews in the evaluation set:", testPositive)

    print("The number of negative reviews in the evaluation set:", testNegative, "\n")

    return trainingData

elif digit == 2:

    return trainingLabels
```

Console Output for review count:

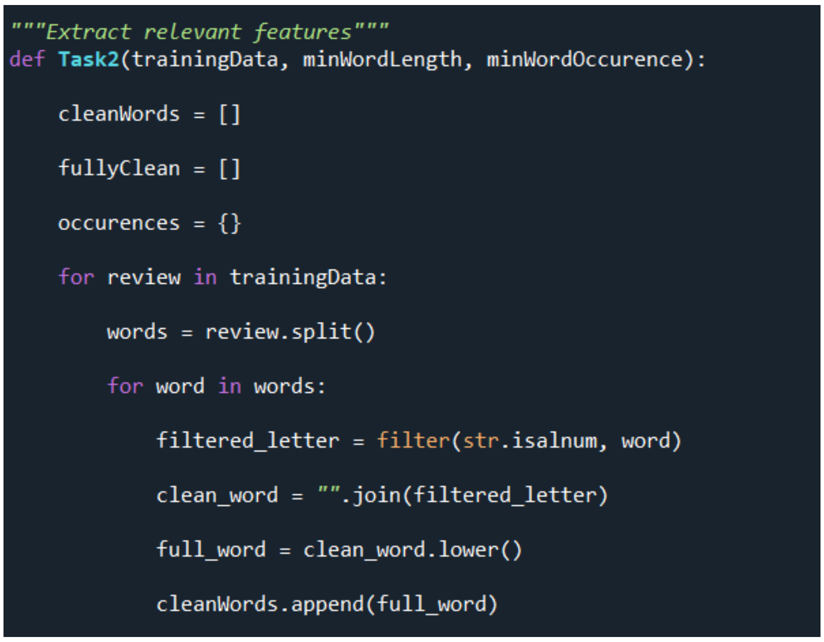
```
The number of positive reviews in the training set: 12500
The number of negative reviews in the training set: 12500
The number of positive reviews in the evaluation set: 12499
The number of negative reviews in the evaluation set: 12500
```

Task 2

In the task the training data is passed by calling Task1 with the integer value of 1 to get the list of training data. Code lines below from 415 to 419.

```
def main():

    trainingData = Task1(1)

    setOfWords = Task2(trainingData, 9, 4000)
```

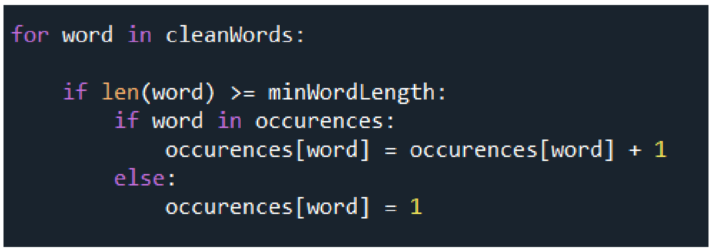
In addition, the minimum word length digit is passed in and the occurrences number of 4000 as displayed above. The next step is to clean the words of unnecessary apostrophe's, dots or any other symbols that are not necessary to be present with a word. This is done, by first splitting each review into separate words. Afterwards, another loop checks each individual letter of a word by using the filter method to make sure the word contains alphanumeric characters. Code lines below from 109 to 130.



```python
"""Extract relevant features"""
def Task2(trainingData, minWordLength, minWordOccurence):

    cleanWords = []

    fullyClean = []

    occurences = {}

    for review in trainingData:

        words = review.split()

        for word in words:

            filtered_letter = filter(str.isalnum, word)

            clean_word = "".join(filtered_letter)

            full_word = clean_word.lower()

            cleanWords.append(full_word)
```

The correct version of the letters are joined together to the variable "clean_word" and then to make sure the word is lowercase, the function lower() is used. So, "full_word" becomes the lowercase version of "clean_word". Each correctly formatted word is added to the list known as "cleanWords" as seen above.

Afterwards, we need to check the length of each word. So we traverse the "cleanWords" list for every word. If the length of the word is greater than or equal to the minimum length than we add the word to the dictionary known as "occurrences" with the value of its occurrence. If the same word appears more than once in the loop, we simply add an additional value to the word in the dictionary. Code lines below from 132 to 138.



```python
for word in cleanWords:

    if len(word) >= minWordLength:
        if word in occurences:
            occurences[word] = occurences[word] + 1
        else:
            occurences[word] = 1
```

Next, we check the occurrence of each word in the dictionary, by comparing the value of each word with the value of minimum word occurrence. If the requirements are met, the word is added to the final list known as "fullyClean". This list is returned in the Task2 method as the "setOfWords" for the classifier. Code lines below from 140 to 144.

```
for word in occurences:
    if occurences[word] >= minWordOccurence:
        fullyClean.append(word)

return fullyClean
```

Task 3

In the Task 3 method, either a positive training review set is passed in or a negative training review. The set of words is also passed into the method from task 2. Depending on the type of review set is passed in, each review of that set is split into words and every word is split into letters to make sure characters there are no non-alphanumeric characters.

Afterwards, each word in that new list of words is traversed through using a for loop. The loop checks if each word of a particular review is contained in the set of words passed in. If this is the case, the "wordFrequency" dictionary will be appended with the word and its value of frequency. If it occurs more than once, an additional value is added to the word in the dictionary. All the words that are not found in the "setOfWords" is given the value of 0 in the dictionary. Finally, the dictionary containing the frequency of each word for a particular review is returned by the Task3 method. Code lines below from 167 to 181.

```
for w in wordList:

    for word in setOfWords:

        if word == w:

            if word in wordFrequency:
                wordFrequency[word] = wordFrequency[word] + 1
            else:
                wordFrequency[word] = 1
        else:
            if word not in wordFrequency:
                wordFrequency[word] = 0

return wordFrequency
```

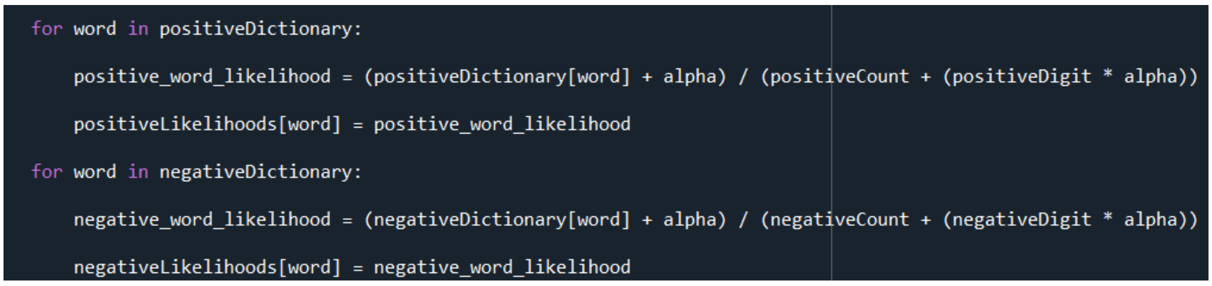
There is a different dictionary for either a positive review or a negative review set. The type of review that we want is retrieved from Task1 by passing the necessary integer value. Code lines below from 421 to 427.

```
positive_review_training_set = Task1(5)

negative_review_training_set = Task1(6)

positive_dictionary = Task3(positive_review_training_set, setOfWords)

negative_dictionary = Task3(negative_review_training_set, setOfWords)
```

We now have the word frequencies for both the positive reviews and the negative reviews, and these dictionaries will be used in Task 4.

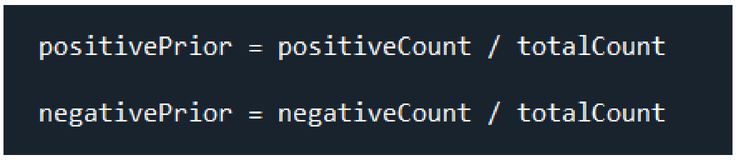
Task 4

To calculate the likelihoods of each word in the positive and negative reviews, we need to pass in the positive and the negative dictionary. For each word of the dictionary, the frequency is retrieved for the word and Laplace smoothing is added by using the alpha value of 1. This is divided by the positive count on which Laplace smoothing is also applied by alpha. The likelihoods for each word are added to the likelihoods dictionary. There is a likelihood word dictionary for both the positive and negative reviews. Code below from lines 200 to 210.



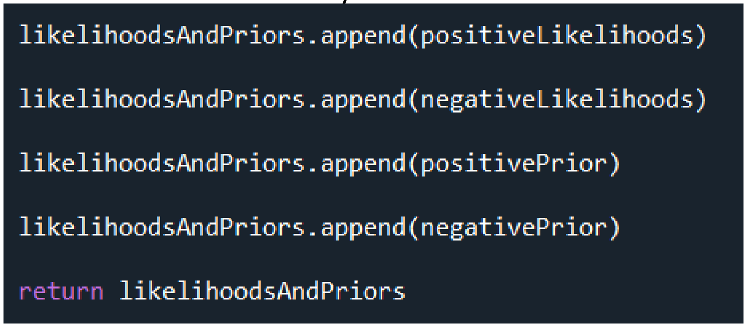
```
for word in positiveDictionary:
    positive_word_likelihood = (positiveDictionary[word] + alpha) / (positiveCount + (positiveDigit * alpha))
    positiveLikelihoods[word] = positive_word_likelihood
for word in negativeDictionary:
    negative_word_likelihood = (negativeDictionary[word] + alpha) / (negativeCount + (negativeDigit * alpha))
    negativeLikelihoods[word] = negative_word_likelihood
```

The calculation of positive priors and negative priors is calculated by dividing the positive and the negative review count by the total count. Code below from lines 212 to 214.



```
positivePrior = positiveCount / totalCount
negativePrior = negativeCount / totalCount
```

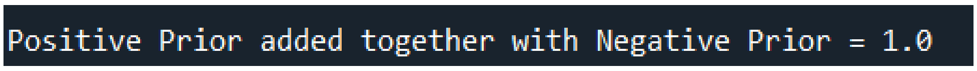
The likelihoods and the priors calculated are then added to the “likelihoodsAndPriors” list. The list is then returned by the Task4 method. Code below from lines 216 to 224.



```
likelihoodsAndPriors.append(positiveLikelihoods)
likelihoodsAndPriors.append(negativeLikelihoods)
likelihoodsAndPriors.append(positivePrior)
likelihoodsAndPriors.append(negativePrior)
return likelihoodsAndPriors
```

A simple test to check if the priors are correct is to add the positive and negative priors together and see if its equal to 1. As seen below, this was truly the case.

Console Output:



```
Positive Prior added together with Negative Prior = 1.0
```

Task5

In this task, the prediction/result is returned from the classifier with 1 being the review is positive and 0 being that the review is negative. However, before this result can be achieved we must pass into the method a single review text, together with the positive likelihoods, negative likelihoods, positive prior and negative prior. The new review that has been not seen by the classifier, as this is new data is filtered just like before to make sure all characters are alphanumeric. Code below from lines 241 to 249.

```
newReviewWords = reviewText.split()

for words in newReviewWords:

    filtered_letter = filter(str.isalnum, words)

    clean_word = "".join(filtered_letter)

    full_word = clean_word.lower()

    cleanWords.append(full_word)
```

Afterwards, we check if every word of the new review exists in either the positive or negative likelihoods dictionaries. If it does, we then extract the likelihood value from that dictionary and convert it to log format by the function math.log(). The positive review log likelihoods are then added to the “logPositive_likelihood” as a total value. This is then converted to an exponent by math.exp() function and stored in the variable “positive_likelihood”. The same procedure is done for the negative likelihoods. The code below is from lines 251 to 267.

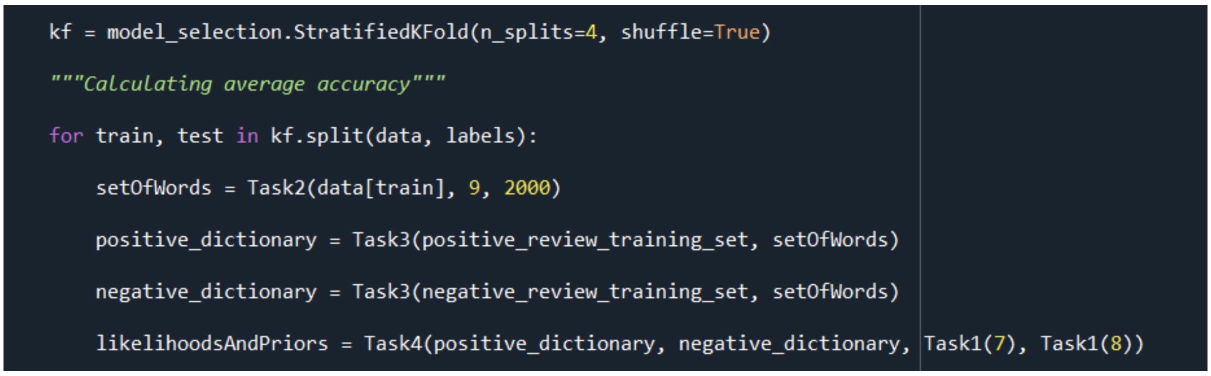
```
for word in cleanWords:

    for positiveWord in positiveLikelihoods:

        if word == positiveWord:
            logPositive_likeLihood = logPositive_likeLihood + math.log(positiveLikelihoods[positiveWord])

positive_likeLihood = math.exp(logPositive_likeLihood)

for word in cleanWords:

    for negativeWord in negativeLikelihoods:

        if word == negativeWord:
            logNegative_likeLihood = logNegative_likeLihood + math.log(negativeLikelihoods[negativeWord])

negative_likeLihood = math.exp(logNegative_likeLihood)
```

Afterwards, the result is obtained depending on the value retrieved from the formula used below. The code exists from lines 267 to 275.

```
if (positive_likeLihood / negative_likeLihood) > (negativePrior / positivePrior):

    """Positive review result"""
    prediction.append(1)
else:
    """Negative review result"""
    prediction.append(0)

return prediction
```

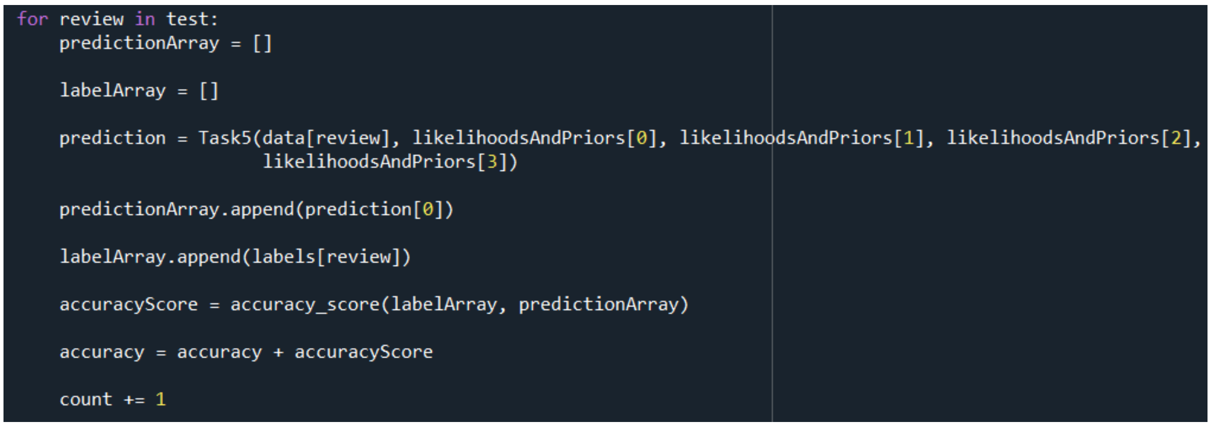
Task 6

In Task 6 the classifier is trained on the training set using the k fold split. The data is the training set and the labels is the training labels set. Code below from lines 306 to 314.



```
kf = model_selection.StratifiedKFold(n_splits=4, shuffle=True)

"""Calculating average accuracy"""

for train, test in kf.split(data, labels):

    setOfWords = Task2(data[train], 9, 2000)

    positive_dictionary = Task3(positive_review_training_set, setOfWords)

    negative_dictionary = Task3(negative_review_training_set, setOfWords)

    likelihoodsAndPriors = Task4(positive_dictionary, negative_dictionary, Task1(7), Task1(8))
```

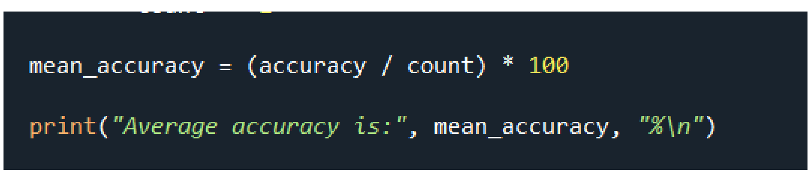
The trained classifier is then evaluated on the evaluation subset by calling Task5 to get predictions and calculating the average accuracy score from the predictions. Code below from lines 320 to 336.



```
for review in test:
    predictionArray = []

    labelArray = []

    prediction = Task5(data[review], likelihoodsAndPriors[0], likelihoodsAndPriors[1], likelihoodsAndPriors[2],
                       likelihoodsAndPriors[3])

    predictionArray.append(prediction[0])

    labelArray.append(labels[review])

    accuracyScore = accuracy_score(labelArray, predictionArray)

    accuracy = accuracy + accuracyScore

    count += 1
```

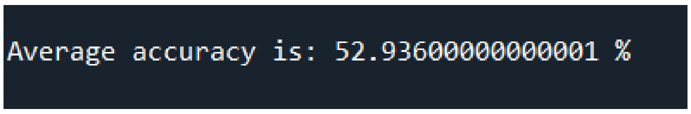
The prediction is retrieved by passing the arguments into Task 5 in the following order, the review text, positive likelihoods by accessing index 0 of the "likelihoodsAndPriors" list. In addition, negative likelihoods by index 1, positive prior by index 2 and negative prior by index 3.

The prediction result is appended to a "predictionArray" list. A particular label based on the index of the review in the for loop is added to the "labelArray" so that this target can be checked against the predicted value when calculating the average. All the average scores are summed up and the average is calculated at the end. Code below from lines 338 to 340.



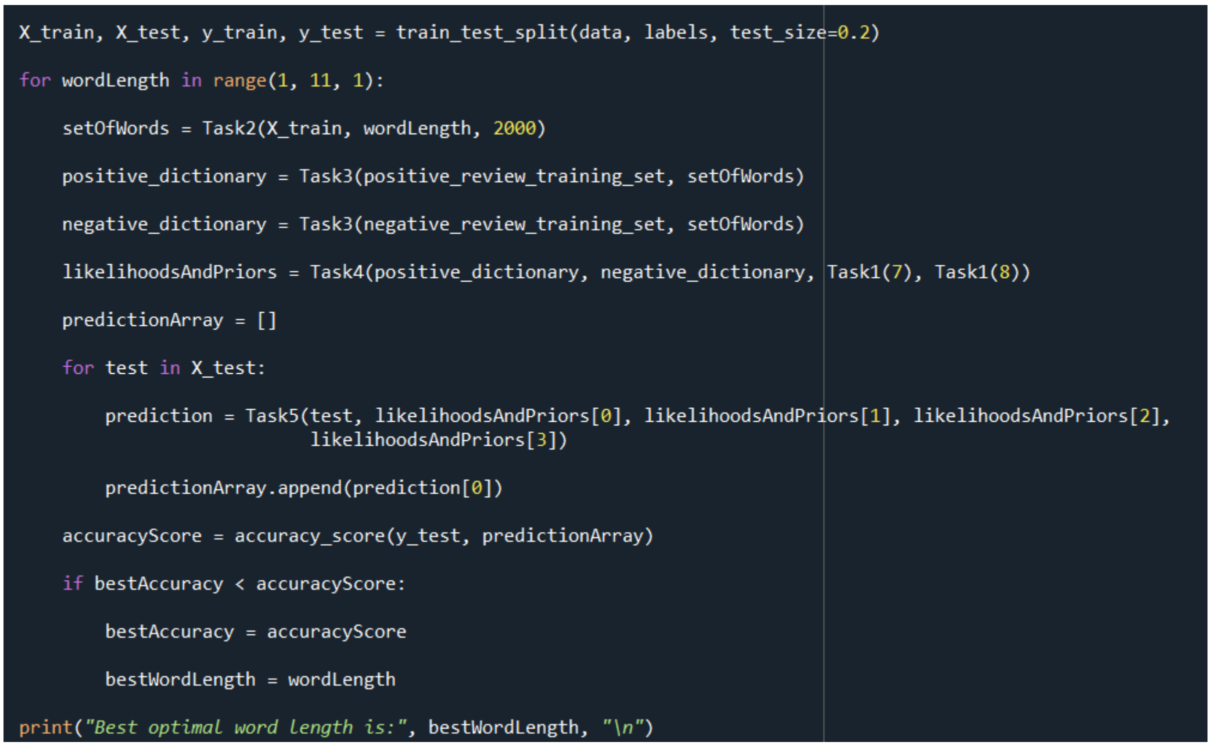
```
mean_accuracy = (accuracy / count) * 100

print("Average accuracy is:", mean_accuracy, "%\n")
```

Console Output:



```
Average accuracy is: 52.936000000000001 %
```

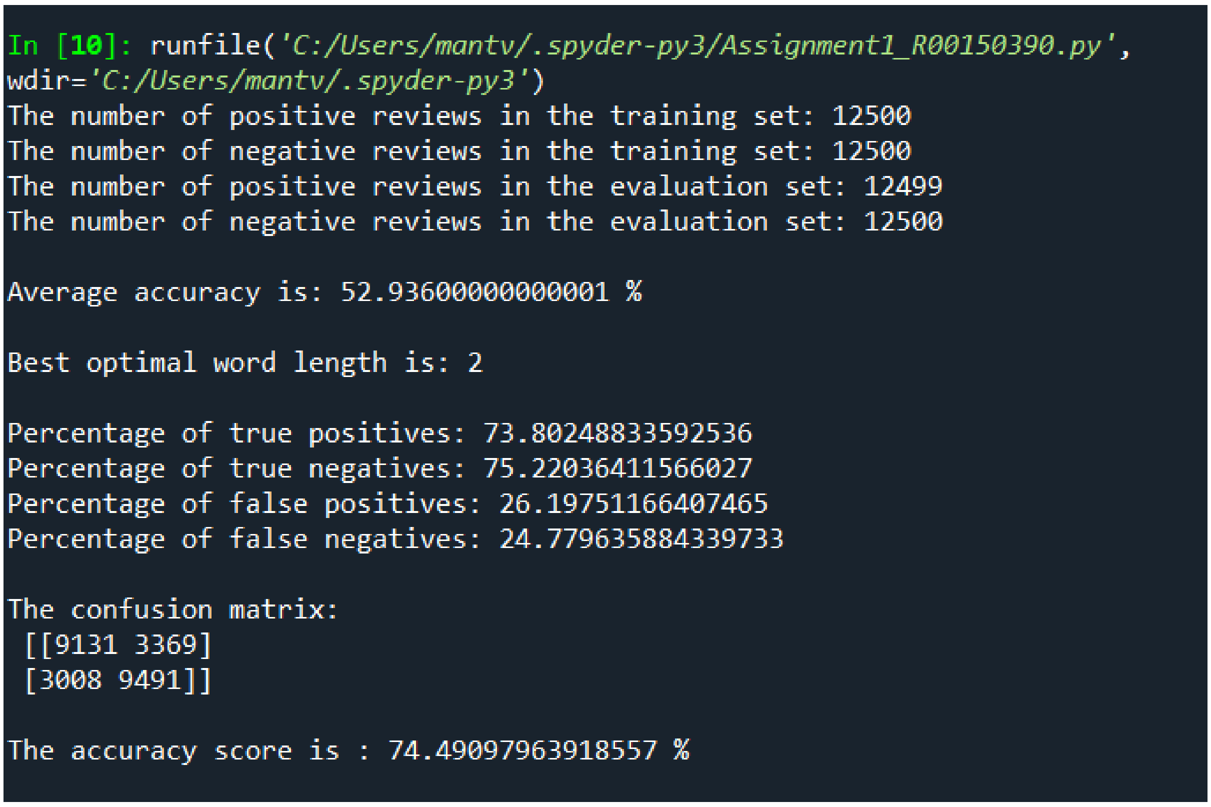
Afterwards, the optimal word length is calculated. For this the training data was split up using train_test_split to speed up the processing on the console. Using the k fold method slowed down the console too much to process the output. The training data was split so that a portion of the reviews can be used as new review data that was not seen before. The classifier was evaluated on the labels. The word length that resulted in largest accuracy score was chosen to be the best word length. The code below is from lines 348 to 377.



```python
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2)

for wordLength in range(1, 11, 1):

    setOfWords = Task2(X_train, wordLength, 2000)

    positive_dictionary = Task3(positive_review_training_set, setOfWords)

    negative_dictionary = Task3(negative_review_training_set, setOfWords)

    likelihoodsAndPriors = Task4(positive_dictionary, negative_dictionary, Task1(7), Task1(8))

    predictionArray = []

    for test in X_test:

        prediction = Task5(test, likelihoodsAndPriors[0], likelihoodsAndPriors[1], likelihoodsAndPriors[2],
                           likelihoodsAndPriors[3])

        predictionArray.append(prediction[0])

    accuracyScore = accuracy_score(y_test, predictionArray)

    if bestAccuracy < accuracyScore:

        bestAccuracy = accuracyScore

        bestWordLength = wordLength

print("Best optimal word length is:", bestWordLength, "\n")
```

Lastly, the classifier was trained once again with the best optimal word length and evaluated on the evaluation set retrieved from task 1. The confusion matrix, true positives, true negatives, false positives, false negatives, and the accuracy score were calculated.

Console Output:



```
In [10]: runfile('C:/Users/mantv/.spyder-py3/Assignment1_R00150390.py',
wdir='C:/Users/mantv/.spyder-py3')
The number of positive reviews in the training set: 12500
The number of negative reviews in the training set: 12500
The number of positive reviews in the evaluation set: 12499
The number of negative reviews in the evaluation set: 12500

Average accuracy is: 52.93600000000001 %

Best optimal word length is: 2

Percentage of true positives: 73.80248833592536
Percentage of true negatives: 75.22036411566027
Percentage of false positives: 26.19751166407465
Percentage of false negatives: 24.779635884339733

The confusion matrix:
 [[9131 3369]
 [3008 9491]]

The accuracy score is : 74.49097963918557 %
```