Quentin JONNEAUX
Student Number : R00274704
COMP 8043 - MACHINE LEARNING

'ASSIGNMENT 1 - BAYESIAN CLASSIFICATION

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

Table of content

Summary....3

Required libraries and functions....4

Task 1 - Splitting and counting reviews....5

Task 2 - Extract relevant features....6

Task 3 - Count feature frequencies....7

Task 4 - Calculate feature likelihoods....8

Task 5 - Maximum likelihood classification....9

Task 6 - Evaluation of results....10

Task 7 - Try classifier on new reviews....12

Appendix 1 - split_count function....13

Appendix 2 - extract_features function....14

Appendix 3 - count_frequencies function....15

Appendix 4 - calc_likelihood_prior function....16

Appendix 5 - classify_max_likelihood function....17

Appendix 6 - K-fold cross-validation function....18

Appendix 7 - Final evaluation function....19

Appendix 8 - New reviews classification function....20

Summary

The purpose of this assignment deals with the creation of a Bayesian Classification algorithm and demonstrate the understanding of the algorithm code and how the classifier function.

The Excel file “movie_reviews.xlsx” on Canvas contains movie reviews from IMDb along with their associated binary sentiment polarity labels (“positive” or ”negative”) and a split indicator to determine if we are supposed to use the corresponding review for training your classifier (“train”) or for the evaluation (“test”).

The creation of the classifier is broken into **6 tasks**. Those 6 tasks involves:
- **Reading and Manipulating the data** contained in the Excel file to create Training data, Training Labels, Test data and Test labels. The function related also display the number of positive and negative reviews in each set (Training and Evaluation)
- **Extracting relevant features** by identifying most important words to train classifier and pre-process the words by applying transformations (removing non-alphanumeric characters, lowercasing)
- **Counting feature frequencies** according to parameters of word length and minimum number of occurrences for likelihoods and priors calculations
- **Classify review** according using logarithmic calculation of likelihoods and priors to perform a Minimum-error-rate classification
- **Evaluate the results** using a k-fold cross-validation procedure, hyper-parameter tuning to maximize model accuracy, and assess accuracy on the test set.

After reading the data, we can see we are dealing with **49,999 reviews** labelled with individual sentiment. Reviews are split between positive and negative reviews as well as a train/test split. Especially:
- 12500 positive reviews are present in the training set
- 12500 negative reviews are present in the training set
- 12499 positive reviews are present in the evaluation set
- 12500 negative reviews are present in the evaluation set

For a minimum number of word occurrence in the training set of **2000**, the best parameter for the minimum word length to be **4**. After final evaluation on the test set, we managed to output a confusion matrix of:
[11064 1436]
[6001 6498]

From the above matrix, we could the following percentages and accuracy score:
- 25.99 % of True positives
- 44.26 % of True negatives
- 5.74 % of False positives
- 24.01 % of False negatives
- Accuracy score of 70.25 %

We also test our classifier, using the best parameters, on newly created reviews. The new positive review has been classified as "positive" and the new negative review has been classified as "negative", which prove the classifier working.

Required libraries and functions

Before developing and running the tasks, we needed to import the required libraries and functions. Especially we needed to import:

- **Pandas** to read the data from the Excel file and manipulate it so we can filter and create subset based on the Sentiment column (Positive and Negative labels) and the Split column (Train vs Test)
- **Numpy** to perform mathematical calculation and manipulating arrays to facilitate cross-validation and evaluation of the classifier
- **Regular expression** for cleaning and pre-processing the review to make them "train" prepared
- **Log from Math** library as logarithmic probabilities will be used to classify reviews while implementing mathematical stability
- **Metrics from Sci-kit Learn** to compute confusion matrices and accuracy scores during the cross validation and evaluation processes
- **Mode Selection from Sci-kit Learn** to facilitate the K-Fold cross validation process

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Oct 10 20:54:14 2025

@author: Quentin
"""

# Importing necessary libraries
import pandas as pd # Pandas for data reading and manipulation
import numpy as np # Numpy for mathematical calculation and manipulating arrays
import re # Regular expression for pre-processing
from math import log # Logarithm for working likelihoods
from sklearn import metrics # Scikit Learn metrics for computing confusion matrices and accuracy scores
from sklearn import model_selection # Scikit Learn model_selection to apply K-Fold cross validation

```

Task 1 - Splitting and counting reviews

The purpose of this task deals with creating the **split_count function**, which is reading the Excel file, splitting the datasets into subsets and counting the number of reviews in different subsets.

We are declaring global variables to facilitate reuse of those outside of the scope of the function, especially to calculate the likelihoods using the number of positive and negative reviews in the training set.

At first, we are creating the data frame by reading the excel file. We then make subset to have series respectively, training review texts, training review sentiments, test review texts and test review sentiments. We are converting those series to Numpy arrays so we can facilitate the computation of confusion matrices and accuracy scores at the cross-validation and evaluation steps.

We also compute numbers of positive and negative reviews in the training set, using the "len" function and pandas filtering features. We are storing those as global variables as those numbers are needed to compute likelihoods used in the classification step, which is outside of the split_count function scope. We also compute the numbers of positive and negative reviews in the evaluation set. We finally output those numbers into the console for visibility and return the created lists (training review texts, training sentiment labels, test review texts, test sentiment labels).

The split_count function definition and a sample output can viewed in Appendix 1.

Task 2 - Extract relevant features

The purpose of this task deals with the creation of the **extract_features function** preprocessing of review texts to make them fit for the algorithm, as non-alphanumeric characters and capitalization would generate variations of the same words, which would affect the training of our classifier. After pre-processing, we extract the words in the whole training set and count their number of appearance. We make a list of the words that is respecting criteria.

We are passing those criteria as input parameters of the function, to return the list of words having a minimum length (minimum number of letters) and minimum number of occurrences in the whole training set.

After declaring a list to store all the words in the training set, we loop through every text in the training reviews. For each text, we apply a regular expression to first replace all non-alphanumeric characters with a space, to avoid connecting unintentionally two words with each other, which would result with non-natural instances that would be used to train the classifier. We then apply another regular expression to replace double or triple spaces, generated by previous step to finish cleaning the data. We then lowercase the word and append it to the list of all words.

After declaring a dictionary that would map each word to its number of occurrences in the training set, we loop through the list of all words and create a dictionary entry for those meeting the minimum word length criteria (namely the number of letter of the word). The code increase the value by 1 for the word each time it is encountered in the list and create the first instance on the first encounter. The dictionary then contains the words as keys and counts of occurrences as values.

Finally, after declaring the main training words list, we loop through the above dictionary to append each word meeting the minimum number of occurrences criteria.

We then return this main word list for the next tasks, as these words will used to count the frequencies and calculate likelihoods of sentiments. The extract_features function definition and a sample output can viewed in Appendix 2.

Task 3 - Count feature frequencies

The purpose of this task deals with the creation of the **count_frequencies function** computing the occurrences of the words in the list returned by the previous task. The function uses the main words list and the training reviews as parameters and will return 2 dictionaries mapping the counts of word presence in positive reviews and in negative reviews. The counts will be used to calculate the likelihoods.

Using the global variables training_labels, we extract the indexes of positive reviews and use them as filter on the training data to create a subset containing only positive review texts. We do the same step to create a subset of negative review texts.

After declaring dictionaries to map word occurrences in both positive and negative subsets, we iterate through each of the main word of the list. For each review text in the positive subset, if the word is present in a positive review, then we create it key with a value of 1 on first encounter and increase the value by one for each review where the word is present. If the word is not present in the subset, we then map it in the dictionary with a value of 0.

We repeat the steps above for the negative review subset. Each dictionary then contains the counts of words appearing at least once in each review of their subset. We then return the dictionaries for the next tasks.

The count_frequencies function definition and a sample output can viewed in Appendix 3.

Task 4 - Calculate feature likelihoods

The purpose of this task deals with the creation of the **calc_likelihood_prior function** computing the likelihoods of word being present given review being positive and also given review being negative. The function also computes the priors, meaning the probabilities that review is positive and also that review is negative. The function takes the dictionaries of word occurrences in positive and negative subsets (from previous tasks) and the counts of positive and negative reviews (declared global in task 1).

We compute the priors by dividing the number of positive reviews in the train set by the total number of reviews. We repeat the same steps to compute the negative prior. As task 1 displayed equal proportions for both class (12500 of positive and negative reviews), both priors equal 0.5.

We then declare a Laplace smoothing parameter alpha and a likelihood dictionary that will map positive and negative likelihoods of word being present in the review given the review class (positive or negative). The alpha parameter to prevent 0 probabilities as this would send the strong statement that a word would never be present in a review given its class, which would not contribute to training the classifier accurately.

We iterate then through words in the positive word occurrences dictionary and compute the number of time each word appear in a positive review. We then compute the likelihood of the word being present given positive class using the following formula:

P[word is present | review is positive] =
(Number of times word appear in positive review x Laplace alpha)
/
(Number of positive review in train subset + Number of Feature in subset x Laplace alpha)

Since the subset contains 2 features (review text and sentiment labels) the alpha is multiplied by 2 in the denominator.

We repeat the same step for negative reviews (since both word occurrences dictionaries contains the same keys, we can perform the computation for negative likelihood in the same loop). We finally map the words with their respective likelihoods to the likelihood dictionary where values are lists including both likelihoods. We then return the likelihood dictionary for the next task.

The calc_likelihood_prior function definition and a sample output can viewed in Appendix 4.

Task 5 - Maximum likelihood classification

The purpose of this task deals with the creation of the **classify_max_likelihood function** computing a sentiment classification for any review, based on the likelihoods calculated in the previous tasks. The functions takes the likelihood dictionaries and the priors returned by the previous task, as well as a review text to classify.

For mathematical stability, we are declaring logarithmic likelihoods for positive and negative sentiments, starting at 0.

We then iterate through every word of the likelihood dictionary. If the word is present in the review text to classify, we store respective likelihoods (first index being for positive review and second for negative reviews). Their logarithmic probabilities are added to the declaration at each word encounter until loop is fully iterated. We produce then logarithmic likelihoods for review being positive and review being negative.

At that stage, we we have computed all information necessary to make a prediction for classifying the review sentiment. We are using the following formula in a if-else statement:

log P[word is present | review is positive] - log log P[word is present | review is negative]
> log P(review is negative) - log P(review is positive)

If the condition above is true, then we make a positive prediction, else we make a negative prediction. We finally return this prediction.

The classify_max_likelihood function definition and a sample output can viewed in Appendix 5.

Task 6 - Evaluation of results

K-fold cross-validation procedure

The purpose of this task deals with the creation of the main function computing the **cross-validation** of the classifier training and its evaluation on the test set, as first step.

The first step deals with executing Task to output the numbers of positive and negative reviews in each set in the console, as well as storing training review texts and labels for cross validation, and test review texts and labels for the final evaluation.

We then create an instance of K-Fold to create a cross-validation process. We choose a number of folds for splitting the training data of 5, as it seems a good compromise for a 49,999 reviews dataset. Indeed, it would make each fold 10,000 reviews (except one fold with 9,999), so enough review to train and validate with while still provide an acceptable response time on the machine, as the cross-validation process is time consuming (approximately 20 minutes). On a more powerful machine, we could have made 10 splits eventually. We are shuffling the indexes to randomize the reviews on each fold and we set the random state as 42 for reproducibility (so that the randomness be the exact same after each run of the script, 42 being completely arbitrary but keep track of randomness).

As we want to pick the best parameter of word length for the evaluation, we declare a mean accuracy dictionary that would map each parameter to its mean accuracy. We are then looping through the range 1 to 10 (inclusive), the iterator i being the word length parameter. We are declaring a list to retain each accuracy in the cross-validation process.

We then conduct the cross-validation process. For each training and test index in the 5-folds split, we first execute task 1 (get a training word list) using the minimum length of i (looping parameter) and the minimum word occurrence parameter of 2000 on the train reviews, using the training indexes produced by the 5-fold split. We are then executing task 2 to count the frequencies of the training split, and task 3 to get its respective likelihoods. At that stage, we have a trained classifier on the intended split data.

We are then computing the validation step. After declaring a list to store the classifier predictions, we are iterating through each review on the test set to make sentiment prediction on each text and append the prediction to the list. By comparing each prediction to its actual outcome in the test labels, we are able to produce a confusion matrix and an accuracy score for each iteration of the split, which we append the accuracies list. Using the list, we can therefore compute a mean accuracy. In the mean accuracies, we can then map each parameter to its mean accuracy.

With a dictionary mapping each parameter for word length to its respective mean accuracy, we can extract the best parameter have the highest accuracy per cross-validation and use it on the final evaluation.

The procedure in the main function definition and a sample output can viewed in Appendix 6.

Final evaluation

We can use the best parameter, output by the cross-validation process, to conduct the **final evaluation**.

We get a main word list by performing task 1 on the whole training set, using the best parameter of word length and a word occurrence parameter of 2000. We then perform task 2 and 3 to calculate the word occurrences in training reviews and compute the likelihoods of the words. At that stage, the classifier is trained with its best parameters and ready for evaluation.

After declaring a list to store the prediction of the test set, we iterate through each review in the test set, and execute task 5 to predict a sentiment label. We append each prediction to the test prediction list.

With this list, we are able to compute a confusion matrix for evaluating the classifier, output its different fractions and calculate the test accuracy score of the classifier. The final evaluation outputs:
- 11064 true negatives (44.26%)
- 6498 true positives (25.99%)
- 1436 false positives (5.74%)
- 6001 false negatives (24.01%)
- 70.25% test accuracy score

The final evaluation indicates that the classifier is accurate more than 2/3 of the time. However, we can see that the classifier is much better on classifying actual negative reviews (5.74% of false positives for 44.26%), as predictions are correct almost 90% of the time. While classifier is less accurate on classifying actual positive reviews (24.01% of false negatives for 25.99% of true positives, as predictions are correct about 50% of the time.

The final evaluation in the main function definition and a sample output can viewed in Appendix 7.

Task 7 - Try classifier on new reviews

As last optional step in the main function, we are curious of **testing the classifier** on new data. We came up with one positive review and one negative review to see if the prediction match the actual sentiment. Here are the new reviews:

- **Positive review (The Green Mile):**

"The Green Mile" is a deeply moving and unforgettable cinematic experience. Tom Hanks shines as Paul Edgecomb, a prison guard on death row, whose life is profoundly changed by John Coffey, played with remarkable sensitivity by Michael Clarke Duncan. Coffey, a gentle giant wrongly convicted of a heinous crime, possesses a mysterious and miraculous gift. The film masterfully blends elements of drama, fantasy, and suspense, creating a compelling narrative that explores themes of faith, redemption, and the inherent value of human life. The supporting cast delivers equally strong performances, adding depth and authenticity to this emotionally resonant story.

More than just a prison drama, "The Green Mile" is a powerful testament to the human spirit's capacity for compassion and empathy, even in the darkest of circumstances. It's a film that stays with you long after the credits roll, prompting reflection on the complexities of morality and the possibility of miracles. While the subject matter is heavy, the film is ultimately uplifting, reminding us to look beyond appearances and recognize the humanity in everyone. A truly exceptional and emotionally rewarding movie.

- **Negative review (Nocturnal Animals):**

"Nocturnal Animals" is a visually stunning but ultimately hollow and pretentious film that prioritizes style over substance. The narrative, split between a wealthy art gallery owner's present and the violent manuscript written by her ex-husband, feels disjointed and emotionally manipulative. The thriller sequences are gratuitously graphic, serving more as shock value than contributing meaningfully to the film's themes. While Amy Adams and Jake Gyllenhaal deliver solid performances, their characters remain largely unsympathetic and underdeveloped, trapped in a cycle of regret and trauma that's difficult to connect with.

The film's biggest flaw is its self-indulgent tone and heavy-handed symbolism. It tries too hard to be clever and provocative, resulting in a convoluted plot and uneven pacing. The connection between the two storylines feels forced and the ending leaves a lingering sense of emptiness rather than genuine emotional resonance. Ultimately, "Nocturnal Animals" is a visually impressive but ultimately unsatisfying experience, leaving the viewer feeling more frustrated than moved. It's a case of style over substance that fails to deliver on its initial promise.

We therefore perform task 5 on both reviews on the new reviews and output their respective prediction in the console. We can see, in both cases, that the classifier was accurate predicting the review on "The Green Mile" as positive and the review on "Nocturnal Animals" as negative.

This step in the main function definition and a sample output can viewed in Appendix 8.

Appendix 1 - split_count function definition and output

Definition:

```
##########################################################################
# Task1 - Splitting and counting the reviews

def split_count():

    # Reading the data and store it in a dataframe (Global variable for reuse in other functions)
    global movies
    movies = pd.read_excel('/Users/Quentin/Desktop/Hdip Data Science and Analytics/Year 1/Semester 3/COMP8043 - Machine Learning/Assignment 1/movie_reviews.xlsx',sheet_name='Sheet1')

    # Subsetting data with training indicator and store training reviews (Global variable for reuse in other functions)
    global training_data
    training_data = movies[movies['Split'] == 'train']['Review'].to_numpy() # Converting to Numpy array to compute metrics

    # Subsetting data with training indicator and store training labels (Global variable for reuse in other functions)
    global training_labels
    training_labels = movies[movies['Split'] == 'train']['Sentiment'].to_numpy() # Converting to Numpy array to compute metrics


    # Subsetting data with test indicator and store test reviews (Global variable for reuse in other functions)
    global test_data
    test_data = movies[movies['Split'] == 'test']['Review'].to_numpy() # Converting to Numpy array to compute metrics

    # Subsetting data with test indicator and store test labels (Global variable for reuse in other functions)
    global test_labels
    test_labels = movies[movies['Split'] == 'test']['Sentiment'].to_numpy() # Converting to Numpy array to compute metrics

    # Storing number of positive review in the training set (Global variable for reuse in other functions)
    global nb_pos_reviews_train
    nb_pos_reviews_train = len(movies[(movies['Split'] == 'train') & (movies['Sentiment'] == 'positive')])

    # Storing number of negative review in the training set (Global variable for reuse in other functions)
    global nb_neg_reviews_train
    nb_neg_reviews_train = len(movies[(movies['Split'] == 'train') & (movies['Sentiment'] == 'negative')])


    # Printing number of positive reviews in the training set
    print('Number of positive reviews in training set: ', nb_pos_reviews_train)
    # Printing number of negative reviews in the training set
    print('Number of negative reviews in training set: ', nb_neg_reviews_train)
    # Printing number of positive reviews in the test set
    print('Number of positive reviews in evaluation set: ', len(movies[(movies['Split'] == 'test') & (movies['Sentiment'] == 'positive')]))
    # Printing number of negative reviews in the test set
    print('Number of negative reviews in evaluation set: ', len(movies[(movies['Split'] == 'test') & (movies['Sentiment'] == 'negative')]))

    # Returning training data, training labels, test data, test labels
    return training_data, training_labels, test_data, test_labels

##########################################################################
```

Output:

```
In [8]: %runfile '/Users/Quentin/Desktop/Hdip Data Science and Analytics/Year
1/Semester 3/COMP8043 - Machine Learning/Assignment 1/COMP8043 - MACHINE
LEARNING - Final.py' --wdir
Number of positive reviews in training set:  12500
Number of negative reviews in training set:  12500
Number of positive reviews in evaluation set:  12499
Number of negative reviews in evaluation set:  12500
```

Appendix 2 - extract_features function definition and output

Definition:

```
64  ################################################################################
65  # Task2 - Extract relevant features
66
67  # Input parameters: training data from split_count, minimum length of words, minimum number of word occurences
68  def extract_features(training_data,min_Word_Length,min_Word_Occurence):
69
70      # Declare a list to store all words in training reviews
71      all_word_list = []
72
73      # For each review in training set
74      for text in training_data:
75          # Storing the text
76          original_text = text
77          # Removing all non-alphanumeric characters of the text
78          clean_text = re.sub(r'[^a-zA-Z0-9]', ' ', original_text)
79          # Removing triple spaces resulting from previous cleaning step
80          clean_text = re.sub(' +', ' ', clean_text)
81          # Lowercasing all words in of the text
82          lower_text = clean_text.lower()
83          # Splitting text to get a list of each word
84          review_word_list = lower_text.split()
85          # Extending all word list with above list
86          all_word_list.extend(review_word_list)
87
88      # Declare a dictionnary mapping words to number of occurence in the training set
89      word_Occurences = {}
90
91      # For every word in the training set (using list created in previous steps)
92      for word in all_word_list:
93          # If word length respects minimum length parameter
94          if (len(word)>=min_Word_Length):
95              # If word is present in dictionnary
96              if (word in word_Occurences):
97                  # Increase occurence count of the word by 1
98                  word_Occurences[word] = word_Occurences[word] + 1
99              # If word not present in dictionnary
100             else:
101                 # Create the word key with a value of 1
102                 word_Occurences[word]=1
103
104
105     # Declare a list of words respecting parameters
106     main_training_word_list = []
107
108     # For each word key in above dictionnary
109     for word in word_Occurences:
110         # if word meet minimum occurence parameter
111         if word_Occurences[word]>=min_Word_Occurence:
112             # Optional - Display word and count in console (uncomment next line if needed)
113             print(word + ":" + str(word_Occurences[word]))
114
115             # Append word to list of word respecting parameters
116             main_training_word_list.append(word)
117
118     # Optional - Display list of words respecting parameters in console (uncomment next line if needed)
119     print(main_training_word_list)
120
121     # Returning list of words respecting parameters
122     return main_training_word_list
123
124 ################################################################################
```

Output: Word occurrences and main training words

```
pretty:3664
good:15147
cast:3830
film:40161
original:3378
comedy:3246
anyone:2632
looking:2483
should:5041
actors:4488
here:5771
little:6438
role:3189
played:2588
feel:2951
trying:2473
those:4697
might:2919
well:10668
young:3660
nothing:4291
story:11990
they:22916
things:3688
take:3510
away:2776
even:12655
seems:3619
takes:2192
```

word_Occurences meeting mininum length and number of occurences Sample output

```
['pretty', 'good', 'cast', 'film', 'original', 'comedy', 'anyone', 'looking',
'should', 'actors', 'here', 'little', 'role', 'played', 'feel', 'trying',
'those', 'might', 'well', 'young', 'nothing', 'story', 'they', 'things', 'take
'away', 'even', 'seems', 'takes', 'from', 'that', 'love', 'part', 'movie',
'just', 'make', 'this', 'first', 'made', 'then', 'again', 'where', 'like',
'instead', 'with', 'hard', 'actor', 'think', 'whole', 'such', 'scenes', 'much',
'only', 'really', 'director', 'enough', 'going', 'there', 'actually', 'place',
'scene', 'having', 'because', 'almost', 'music', 'also', 'through', 'have',
'always', 'when', 'still', 'very', 'book', 'were', 'excellent', 'everyone',
'seen', 'movies', 'great', 'something', 'never', 'before', 'idea', 'what',
'about', 'shot', 'funny', 'makes', 'your', 'find', 'would', 'some', 'watch',
'although', 'does', 'thought', 'point', 'every', 'gets', 'their', 'which',
'real', 'people', 'shows', 'these', 'american', 'look', 'being', 'life', 'john
'quite', 'will', 'better', 'fact', 'didn', 'more', 'than', 'probably',
'characters', 'know', 'script', 'plot', 'between', 'while', 'course', 'though',
'into', 'sense', 'different', 'most', 'them', 'since', 'half', 'true', 'both',
'over', 'come', 'show', 'each', 'watching', 'down', 'around', 'times',
'especially', 'character', 'interesting', 'best', 'nice', 'reason', 'give',
'could', 'been', 'star', 'series', 'after', 'right', 'left', 'time', 'seeing',
'same', 'said', 'woman', 'done', 'everything', 'audience', 'acting', 'films',
'wasn', 'thing', 'worst', 'sure', 'girl', 'other', 'goes', 'least', 'another',
'looks', 'play', 'work', 'ever', 'watched', 'years', 'high', 'main', 'maybe',
'minutes', 'action', 'ending', 'last', 'else', 'must', 'seem', 'believe',
'world', 'night', 'year', 'during', 'house', 'many', 'version', 'effects',
'worth', 'kind', 'screen', 'plays', 'however', 'back', 'performance', 'making',
'wife', 'once', 'rather', 'less', 'black', 'himself', 'anything', 'family',
'three', 'special', 'together', 'want', 'beautiful', 'doesn', 'without',
'horror', 'comes', 'father', 'later', 'someone', 'long', 'money', 'found']
```

Appendix 3 - count_frequencies function definition and output

Definition:

```
124  ######################################################################
125  # Task3 - count feature frequencies
126
127  # Input parameters: training data from split_count, list of words respecting parameters from extract_features
128  def count_frequencies(training_data,main_training_word_list):
129
130      # Storing indexes of positive labels in training set
131      train_pos_indexes = np.where([training_labels=='positive'][0])
132      # Storing positive reviews in training set using labels above
133      train_pos_reviews = training_data[train_pos_indexes]
134
135      # Storing indexes of negative labels in training set
136      train_neg_indexes = np.where([training_labels=='negative'][0])
137      # Storing negative reviews in training set using labels above
138      train_neg_reviews = training_data[train_neg_indexes]
139
140      # Declaring dictionaries to count presence of word in positive reviews and negative reviews, respectively
141      pos_word_occurences= {}
142      neg_word_occurences= {}
143
144      # For each word in the word list respecting parameters
145      for word in main_training_word_list:
146          # For each review in all positive reviews in training set
147          for review in train_pos_reviews:
148              # If word is present in review
149              if word in review.lower().split():
150                  # If word present in positive review dictionary
151                  if word in pos_word_occurences:
152                      # Increase occurence count by 1
153                      pos_word_occurences[word] = pos_word_occurences[word] + 1
154                  # If word not present in positive review dictionnary
155                  else:
156                      # Create the word key with a value of 1
157                      pos_word_occurences[word]=1
158          # If word not present in dictionary after all reviews
159          if word not in pos_word_occurences.keys():
160              # Create the word key with a value of 0
161              pos_word_occurences[word]=0
162
163      # For each word in the word list respecting parameters
164      for word in main_training_word_list:
165          # For each review in all negative reviews in training set
166          for review in train_neg_reviews:
167              # If word is present in review
168              if word in review.lower().split():
169                  # If word present in negative review dictionary
170                  if word in neg_word_occurences:
171                      # Increase occurence count by 1
172                      neg_word_occurences[word] = neg_word_occurences[word] + 1
173                  # If word not present in positive review dictionnary
174                  else:
175                      # Create the word key with a value of 1
176                      neg_word_occurences[word]=1
177          # If word not present in dictionary after all reviews
178          if word not in neg_word_occurences.keys():
179              # Create the word key with a value of 0
180              neg_word_occurences[word]=0
181
182      # Optional - Display occurence dictionnaries in console (uncomment next 2 lines if needed)
183      print(pos_word_occurences)
184      print(neg_word_occurences)
185
186      # Retuning word occurence dictionnaries (one for each types)
187      return pos_word_occurences,neg_word_occurences
188
```

Output:

```
{'pretty': 1191, 'original': 891, 'comedy': 874, 'anyone': 951, 'should': 1692
'actors': 1309, 'little': 2402, 'played': 1089, 'those': 1951, 'might': 1005,
'young': 1576, 'nothing': 982, 'story': 3462, 'things': 1332, 'seems': 1238,
'movie': 6066, 'first': 3073, 'again': 936, 'where': 2263, 'think': 2485,
'whole': 1061, 'scenes': 1466, 'really': 3495, 'director': 1218, 'enough': 990
'going': 1389, 'there': 4005, 'actually': 1350, 'scene': 1465, 'because': 2763
'almost': 1287, 'music': 974, 'through': 1758, 'always': 1576, 'still': 2335,
'movies': 1911, 'great': 3786, 'something': 1563, 'never': 2493, 'before': 147
'about': 4895, 'funny': 993, 'makes': 1962, 'would': 3574, 'watch': 2248,
'although': 1136, 'thought': 1234, 'point': 844, 'every': 1610, 'their': 3593,
'which': 3592, 'people': 2606, 'these': 2001, 'being': 2398, 'quite': 1672,
'better': 1605, 'probably': 1153, 'characters': 1977, 'script': 608, 'between'
1535, 'while': 2048, 'though': 1451, 'since': 1192, 'watching': 1454, 'around'
1210, 'times': 994, 'especially': 1230, 'character': 1947, 'interesting': 1005
'could': 2413, 'series': 828, 'after': 2723, 'right': 1104, 'woman': 836,
'acting': 1633, 'films': 1945, 'thing': 1125, 'worst': 195, 'other': 3209,
'least': 877, 'another': 1566, 'years': 1683, 'minutes': 507, 'action': 886,
'world': 1249, 'however': 356, 'performance': 1187, 'making': 920, 'rather':
1050, 'anything': 830, 'family': 1053, 'doesn': 0, 'without': 1409, 'horror':
665, 'comes': 1096, 'found': 1106}
{'pretty': 1635, 'original': 1080, 'comedy': 672, 'anyone': 1055, 'should':
2257, 'actors': 1497, 'little': 2272, 'played': 786, 'those': 1742, 'might':
1382, 'young': 952, 'nothing': 2167, 'story': 2732, 'things': 1306, 'seems':
1588, 'movie': 7251, 'first': 2849, 'again': 633, 'where': 2333, 'think': 2646
'whole': 1441, 'scenes': 1631, 'really': 3872, 'director': 1406, 'enough': 134
'going': 1795, 'there': 4954, 'actually': 1857, 'scene': 1576, 'because': 3361
'almost': 1297, 'music': 755, 'through': 1886, 'always': 930, 'still': 1703,
'movies': 2097, 'great': 1802, 'something': 2037, 'never': 2438, 'before': 150
'about': 5278, 'funny': 1035, 'makes': 1544, 'would': 4377, 'watch': 2411,
'although': 781, 'thought': 1393, 'point': 1137, 'every': 1655, 'their': 3373,
'which': 3456, 'people': 2842, 'these': 2015, 'being': 2521, 'quite': 1318,
'better': 2158, 'probably': 1209, 'characters': 2083, 'script': 1259, 'between
1112, 'while': 1777, 'though': 1185, 'since': 1142, 'watching': 2038, 'around'
1364, 'times': 824, 'especially': 739, 'character': 1959, 'interesting': 1039,
'could': 3373, 'series': 585, 'after': 2859, 'right': 967, 'woman': 782,
'acting': 2513, 'films': 1585, 'thing': 1920, 'worst': 1886, 'other': 3055,
'least': 1536, 'another': 1714, 'years': 1128, 'minutes': 1278, 'action': 823,
'world': 748, 'however': 381, 'performance': 609, 'making': 1241, 'rather':
1123, 'anything': 1354, 'family': 621, 'doesn': 0, 'without': 1332, 'horror':
1094, 'comes': 1031, 'found': 1065}
```

Appendix 4 - calc_likelihood_prior function definition and output

Definition:

```
###############################################################################
# Task4 - calculate feature likelihoods and priors

# Input parameters: word occurence dictionnaries from count_frequencies, number of positive and negative reviews from split_count
def calc_likelihood_prior(pos_word_occurences,neg_word_occurences,nb_pos_reviews_train,nb_neg_reviews_train):

    # Storing prior of positive reviews
    pos_prior = nb_pos_reviews_train/(nb_pos_reviews_train+nb_neg_reviews_train)
    # Optional - Display prior of positive reviews in the console (uncomment next line if needed)
    print('Positive prior: ',pos_prior)

    # Storing prior of negative reviews
    neg_prior = nb_neg_reviews_train/(nb_pos_reviews_train+nb_neg_reviews_train)
    # Optional - Display prior of negative reviews in the console (uncomment next line if needed)
    print('Negative prior: ',neg_prior)

    #  Declare a laplace smoothing parameter of 1
    laplace_alpha = 1

    # Declare a likelihood dictionnary
    likelihood_dict={}

    # For each word in the positive occurence dictionnary
    for word in pos_word_occurences.keys():

        # Storing number of times word is appearing in a positive review
        nb_times_word_in_pos_review = pos_word_occurences[word]
        # Storing likelihood that the word is present in positive review (applying the laplace smoothing)
        likelihood_pos = (nb_times_word_in_pos_review + laplace_alpha)/(nb_pos_reviews_train + 2 * laplace_alpha)

        # Storing number of times word is appearing in a negative review
        nb_times_word_in_neg_review = neg_word_occurences[word]
        # Storing likelihood that the word is present in negative review (applying the laplace smoothing)
        likelihood_neg = (nb_times_word_in_neg_review + laplace_alpha)/(nb_neg_reviews_train + 2 * laplace_alpha)

        # Map the word in the likelihood dictionary to positive and negative likelihoods
        likelihood_dict[word] = [likelihood_pos,likelihood_neg]

    # Optional - Display likelihood dictionary in the console (uncomment next line if needed)
    print(likelihood_dict)

    # Return likelihood dictionary and priors
    return likelihood_dict,pos_prior,neg_prior

###############################################################################################
```

Output:

```
Positive prior:  0.5
Negative prior:  0.5
{'pretty': [0.09534474484082547, 0.1308590625499921, 'good':
[0.32706766917293234, 0.32826747720364741, 'cast': [0.11086226203807391,
0.09062549992001281, 'film': [0.48968165093585025, 0.45792673172292431,
'original': [0.07134858422652375, 0.08646616541353383], 'comedy':
[0.06998880179171332, 0.05383133869780835051, 'anyone': [0.0761478163493841,
0.08446648536234203], 'looking': [0.066309390497520 4, 0.08462645976643737],
'should': [0.13541833306670933, 0.18061110222364421, 'actors':
[0.10478323468245081, 0.11982028866741321], 'here': [0.10398336266197408,
0.10510318349064151, 'little': [0.19220924652055671, 0.18181091025435931, 'r
[0.08718605023196288, 0.05415133578627419], 'played': [0.0871860502319628 8,
0.06294992801151816], 'feel': [0.09614461686130221, 0.08694608862581987],
'trying': [0.06774916013437851, 0.10254359302511598], 'those':
[0.15613501839705646, 0.13941769316909294], 'might': [0.08046712525995841,
0.11062230043193089], 'well': [0.23028315469524877, 0.14037753959366651, 'yo
[0.12613981762917933, 0.07622780355143177], 'nothing': [0.0786274196128619 5,
0.17341225403935371, 'story': [0.27699568069108943, 0.21860502319628861, 't
[0.37010078387458006, 0.44952807550791873], 'things': [0.10662294032954728,
0.10454327307630791, 'take': [0.11190209566469365, 0.11830107182850744],
'away': [0.06902895536714125, 0.07750759878419453], 'even':
[0.26931690929451285, 0.38753799392097266], 'seems': [0.0991041433370660 8,
0.12709966405375141, 'takes': [0.08638617821148616, 0.06462965925451927],
'from': [0.47000479923212285, 0.45456726923692211, 'that': [0.7749960006398
0.80427131658934571, 'love': [0.19892817149256121, 0.11358182690769476], 'pa
[0.11006239001759718, 0.10502319628859383], 'movie': [0.4852823548232283,
0.58006718924972], 'just': [0.36314189729643261, 0.47064469684850424], 'make
[0.20028795392737161, 0.27059670452727563], 'this': [0.8803391457366821,
0.91369380899056151, 'first': [0.24588065909454487, 0.22796352583586627],
'made': [0.20644696848504241, 0.23668213085906256], 'then': [0.1738921772516
0.23260278355463126], 'again': [0.07494800831866902, 0.05071188609822429],
'where': [0.18109102543593025, 0.18669012957926731, 'like': [0.4094544872820
0.48176291793313071, 'instead': [0.040073558822588386, 0.07414813629819231,
'with': [0.7008478643417053, 0.67957126859702441, 'hard': [0.0770276755719
0.08278675411934091, 'actor': [0.055031195008798591, 0.05383138690780835051,
'think': [0.19848418429051351, 0.21172612382018877], 'whole':
[0.08494640857462805, 0.11534154535274356], 'such': [0.15109582466805311,
0.15397536394176931, 'scenes': [0.11734122540393536, 0.13053911374180131,
'much': [0.24788033914573681, 0.27363621820508721, 'only': [0.2902735562310
0.36666133418653013], 'really': [0.27963525835866261, 0.30979043353063511,
```

Appendix 5 - classify_max_likelihood function definition and output

Definition:

```
##########################################################################
# Task5 - maximum likelihood classification

# Input parameters: likelihood dictionary and priors from calc_likelihood_prior, review text to classify
def classify_max_likelihood(likelihood_dict,pos_prior,neg_prior,new_text):

    # Declaring logarithmic likelihoods as 0 (for both positive and negative likelihoods)
    logLikelihood_pos = 0
    logLikelihood_neg = 0

    # For each word in the likelihood dictionary
    for word in likelihood_dict.keys():
        # If word present in the review text to classify
        if word in new_text.split():
            # Store both likelihoods of word
            likelihood_word_pos = likelihood_dict[word][0]
            likelihood_word_neg = likelihood_dict[word][1]

            # Add word logarithmic likelihoods to related class likelihood
            logLikelihood_pos = logLikelihood_pos + log(likelihood_word_pos)
            logLikelihood_neg = logLikelihood_neg + log(likelihood_word_neg)

    # Optional - Display logarithmic likelihoods in console (uncomment next 2 lines if needed)
    print(logLikelihood_pos)
    print(logLikelihood_neg)

    # Minimum-error-rate classification
    # If difference of logarithmic likelihoods (Positive - Negative) greater than difference of logarithmic priors (Negative - Positive)
    if logLikelihood_pos - logLikelihood_neg > log(neg_prior) - log(pos_prior):
        # Predicting review is positive
        prediction = 'positive'
        # Optional - Display positive prediction in console (uncomment next line if needed)
        print('review is:', prediction)

    # If difference of logarithmic likelihoods (Positive - Negative) NOT greater than difference of logarithmic priors (Negative - Positive)
    else:
        # Predicting review is negative
        prediction = 'negative'
        # Optional - Display negative prediction in console (uncomment next line if needed)
        print('review is:', prediction)

    #  Return predicted sentiment label
    return(prediction)

##########################################################################
```

Output:

```
-35.779806417509995
-33.09317700396092
review is: negative
-15.18342776260919
-15.230220156543046
review is: positive
-46.855486704601844
-46.636744022252756
review is: negative
-34.245912938471626
-35.26003415957725
review is: positive
-29.343919853643165
-29.448246769986966
review is: positive
```

Appendix 6 - K-fold cross-validation function definition and output

Definition:

```python
# Declaring a main script to execute
def main():

    # Storing data and labels from training and test sets from split_count (Task1)
    training_data, training_labels, test_data, test_labels = split_count()

    # Creating a K-Fold cross validation procedure (Using 5 folds, shuffling indexes, setting random_state as 42 for reproducibility)
    kf = model_selection.KFold(n_splits=5, shuffle=True, random_state=42)

    # Create a dictionary mapping each minimum word length to mean accuracies
    mean_accuracies = {}

    # For each minimum word length parameter (1,2,3,4,5,6,7,8,9,10)
    for i in range(1,11):

        # Declare a list of accuracies for each subset
        accuracies = []

        # For each train and test indexes in the K-fold procedure split
        for train_index, test_index in kf.split(training_data, training_labels):
                # Get the training word list using indexes used for training by K-Fold procedure, minimum length parameter and an arbitrary minimum word occurence parameters (4000 here) from e
                main_training_word_list = extract_features(training_data[train_index],i,2000)
                # Get positive and negative occurences from count_frequencies (Task 3)
                pos_word_occurences,neg_word_occurences = count_frequencies(training_data,main_training_word_list)
                # Get likelihood dictionary and priors from calc_likelihood_prior (Task 4)
                likelihood_dict,pos_prior,neg_prior = calc_likelihood_prior(pos_word_occurences,neg_word_occurences,nb_pos_reviews_train,nb_neg_reviews_train)

                # Declare a list to store predictions for test subset
                preds = []
                # For each review text in test subset
                for text in training_data[test_index]:
                    # Predict class of review
                    pred = classify_max_likelihood(likelihood_dict, pos_prior, neg_prior,text)
                    # Append prediction to list
                    preds.append(pred)

                # Optionnal - Compute confusion matrix and display in console (uncomment next 2 lines if needed)
                C = metrics.confusion_matrix(training_labels[test_index], preds)
                print(C)

                # Store accuracyy score of classifier
                accuracy = metrics.accuracy_score(training_labels[test_index], preds)
                # Append accuracy score to list of accuracies
                accuracies.append(accuracy)

        # Compute and Store mean of accuracies in the list
        mean_accuracy = np.mean(accuracies)
        # Map parameter with mean accuracy to
        mean_accuracies[i]= mean_accuracy

    # Optional - Display mean accuracies dictionary in the console (uncomment next line if needed)
    print(mean_accuracies)

    # For each key-value pair in mean accuracies dictionary
    for k, v in mean_accuracies.items():
        # if value is highest mean accuracy
        if v == max(mean_accuracies.values()):
            # Store key parameter as best parameter
            best_param = k
            # Leave the loop
            break


    # Optional - Display best parameter in the console (uncomment next line if needed)
    print('Best minimum word length parameter is: ',best_param)
```

Output:

```
[[2297  250]
 [2040  413]]
[[2262  257]
 [2056  425]]
[[2224  248]
 [2086  442]]
[[2366  102]
 [2282  250]]
[[2253  241]
 [2075  431]]
{1: np.float64(0.64876), 2: np.float64(0.65088), 3:
np.float64(0.6769999999999999), 4: np.float64(0.68008), 5:
np.float64(0.6513599999999999), 6: np.float64(0.58036), 7: np.float64(0.55372)
8: np.float64(0.53992), 9: np.float64(0.53452), 10: np.float64(0.53452)}
Best minimum word length parameter is:  4
```

Appendix 7 - Final evaluation function definition and output

Definition:

```
    # Get the training word list using training data, best minimum length parameter and an arbitrary minimum word occurence parameters (2000 here) from extract_features (Task 2)
    main_training_word_list = extract_features(training_data,best_param,2000)
    # Get positive and negative occurences from count_frequencies (Task 3)
    pos_word_occurences,neg_word_occurences = count_frequencies(training_data,main_training_word_list)
    # Get likelihood dictionary and priors from calc_likelihood_prior (Task 4)
    likelihood_dict,pos_prior,neg_prior = calc_likelihood_prior(pos_word_occurences,neg_word_occurences,nb_pos_reviews_train,nb_neg_reviews_train)

    # Declare a list to store predictions for test data
    test_preds = []
    # For each review text in test data
    for text in test_data:
        # Predict class of review
        test_pred = classify_max_likelihood(likelihood_dict, pos_prior, neg_prior,text)
        # Append prediction to list
        test_preds.append(test_pred)

    # Compute confusion matrix for final evaluation
    C = metrics.confusion_matrix(test_labels, test_preds)

    # Storing confusion matrix values
    test_true_negative = C[0,0]
    test_true_positive = C[1,1]
    test_false_negative = C[1,0]
    test_false_positive = C[0,1]

    # Display confusion matrix for the classification in the console
    print(C)

    # Display The percentage of true positives, true negatives, false positives and false negatives in the console
    print("% True positives: ",(test_true_positive/len(test_labels)) * 100, "%")
    print("% True negatives: ",(test_true_negative/len(test_labels)) * 100, "%")
    print("% False positives: ",(test_false_positive/len(test_labels)) * 100, "%")
    print("% False negatives: ",(test_false_negative/len(test_labels)) * 100, "%")

    # Storing the classification accuracy score
    test_accuracy = metrics.accuracy_score(test_labels, test_preds)
    # Display the classification accuracy score in the console
    print("Test accuracy score: ", test_accuracy * 100, "%")
```

Output:

```
[[11064  1436]
 [ 6001  6498]]
% True positives:  25.99303972158886 %
% True negatives:  44.25777031081243 %
% False positives:  5.744229769190767 %
% False negatives:  24.004960198407936 %
Test accuracy score:  70.2508100324013 %
```

Appendix 8 - New reviews classification function definition and output

Definition:

```
##########################################################################
# Task7 - Optional, Trying classifier on newly written reviews

    # Storing a new positive review (Movie: The Green Mile)
    positive_text = "The Green Mile is a deeply moving and unforgettable cinematic experience. Tom Hanks shines as Paul Edgecomb, a prison guard on death row, whose life is profoundly changed by
    # Predicting the sentiment of review
    new_pred = classify_max_likelihood(likelihood_dict, pos_prior, neg_prior,positive_text)
    # Display prediction of new positive review in the console
    print('New review of The Green Mile is predicted: ', new_pred)

    # Storing a new negative review (Movie: Nocturnal Animals)
    negative_text = "Nocturnal Animals is a visually stunning but ultimately hollow and pretentious film that prioritizes style over substance. The narrative, split between a wealthy art gallery
    # Predicting the sentiment of review
    new_pred = classify_max_likelihood(likelihood_dict, pos_prior, neg_prior,negative_text)
    # Display prediction of new negative review in the console
    print('New review of Nocturnal Animals is predicted: ', new_pred)
```

Output:

```
New review of The Green Mile is predicted:  positive
New review of Nocturnal Animals is predicted:  negative
```