2024

Machine Learning Assignment 1

RESULTS AND REPORT

EMMETT FITZHARRIS R00222357

Task 1

I start by reading the data from the excel file in read_data()

```
# Read data from the Excel file.
# The file is large, and this process is slow
# since I will be running this a number of times, I decided to cache the data frame
def read_data(): 1 usage   Emmett *
    cache_path = os.path.join(cache_folder, main_cache_filename)
    if os.path.exists(cache_path):
        with open(cache_path, 'rb') as f:
            df = pickle.load(f)
        print('Loaded data from cache\n')
    else:
        df = pd.read_excel(source_filename + file_extension)
        with open(cache_path, 'wb') as f:
            pickle.dump(df, f)
        print('Loaded data from Excel and cached it')
    return df
```

Knowing that I would be running this program many times, I decided to implement a cache system to serialize the data frame, rather than needing to read from the Excel file each time.

Initial read time was approximately 5.05 seconds

```
Execution time: 5.05 seconds
```

After caching, read time is less than 0.13 seconds.

```
Execution time: 0.13 seconds
```

This allows me to iterate more quickly, rather than waiting for the same data to be read again and again. I recorded this result by simply wrapping my main function as follows:

```python
def main():  1 usage  new *
    start_time = time.time()
    df = read_data()

    print("Frame size", df.shape[0])
    train, test = split_data(df)


    end_time = time.time()
    elapsed_time = end_time - start_time
    print(f"Execution time: {elapsed_time:.2f} seconds")
```

This early stage of the program does not have particularly exciting results, but I continue to cache results across the program, which results in significant time saving.

In the task1() function, passing in the data frame. Which splits the data into test and training sets, then prints the counts of positive vs negative labels.

```
--------------------------------------------------------------------------------

Total row count 49999

--------------------------------------------------------------------------------

Positive Labels in training data: 12500
Negative Labels in training data: 12500

--------------------------------------------------------------------------------

Positive Labels in test data: 12499
Negative Labels in test data: 12500
```

I then bundle the outputs into Pandas data frames and pass back to main.

Task 2

In task two I filter the data frames by minimum_word_length and minimum_word_occurrence.

This is done by filtering out non alphanumeric characters with a pair of reg-ex queries. The data is then made lower case, and each word is split into a list item.

A new data frame is created, but finding those words which meet the above requirements, and running the value count function to count occurrences. This data frame will act as the feature set for model training.

```
[24999 rows x 2 columns]
Number of unique words in training data, longer than 12 characters, which occur more than 100
  times: 71
Number of unique words in test data, longer than 12 characters, which occur more than 100 times: 67
```

The data appears to be evenly split. The result of these queries variers greatly based on the parameters passed into it. The values I chose at this stage were intentionally high to keep iterations quick:

```
minimumWordLength, minimumWordOccurrence = 12, 100 # these values are intentionally high for testing
```

Task 3

In task 3, we count the number of positive / negative reviews each word occurs in. The function I made for this task requires a loop which takes a long time to run, so I implemented a caching system again to store the result. I was mindful to ensure that it was sensitive to changing minimum word length and minimum occurrence parameters.

At a minimum word length of 10 and a minimum word occurrence of 20, the execution time observed was over 80 seconds:

```
Execution time: 83.73 seconds
```

With the cache in place the program takes less that 7 seconds to run. A reduction in runtime of over a minute. Keep in mind that these parameter values are very high for testing, so the difference will be even more pronounced at normal values, making iterations much more quick.

```
Execution time: 6.75 seconds
```

My results below are intentionally using long words to keep runtime low:

```
##############################################################################
 #######
                    Task 3
##############################################################################
 #######

 100.00% | Elapsed: 1.18s | Remaining: 0.00s
Saved data to cache: cache\movie_reviews-train_word_counts_pos-12-100.pkl

            Word  review_count
0    performances          1083
1     entertaining           785
5     relationship           723
4   cinematography           525
3     particularly           502
..             ...           ...
27   uninteresting            25
29    unconvincing            24
61    ridiculously            21
59    unbelievably            21
46  writerdirector             0

[71 rows x 2 columns]
```

Positive reviews rarely contain negative descriptions like “uninteresting, or “unconvincing”.

```
--------------------------------------------------------------------------------

 100.00% | Elapsed: 1.50s | Remaining: 0.00s
Saved data to cache: cache\movie_reviews-train_word_counts_neg-12-100.pkl

             Word  review_count
1     entertaining          555
0     performances          525
6     disappointed          524
3     particularly          431
4   cinematography          374
..             ...          ...
33   extraordinary           25
35    breathtaking           25
58    psychiatrist           24
44   unforgettable           22
46  writerdirector            0

[71 rows x 2 columns]
```

On the opposite side, negative reviews rarely contain positive terms, like “unforgettable, or “extraordinary”.

In both cases, the most frequently used words tend to describe the industry, such as “entertaining”, or “cinematography”.

Task 4

In task 4, I calculate some statistics regarding the data. The probability that a word is in a review, given that it is positive, as well as the reverse, also know as the prior probabilities.

```
##############################################################################
 #######
                    Task 4
##############################################################################
 #######

Priors
Training DataProbability. Positive: 0.50, Negative: 0.50
Test Data Probability. Positive: 0.50, Negative: 0.50

----------------------------------------------------------------------------

 Number of training Reviews: 25000
 Number of test Reviews: 24999

----------------------------------------------------------------------------
```

This turns out to be almost exactly 50/50, which I confirmed by printing the numbers of each type of review.

Following this, I calculate the conditional probabilities, and add them as a

new column in my data frame. The conditional probabilities are calculated as follows:

```
Training Data

Positive
            Word  review_count  probability  conditional_probability
0   performances          1083     0.043357                 0.417343
1    entertaining          785     0.031437                 0.412575
2   unfortunately          187     0.007519                 0.403008
3    particularly          502     0.020118                 0.408047
4  cinematography          525     0.021038                 0.408415
..            ...          ...          ...                      ...
66   deliberately           56     0.002280                 0.400912
67  unintentional           39     0.001600                 0.400640
68   consistently           62     0.002520                 0.401008
69  revolutionary           46     0.001880                 0.400752
70   specifically           46     0.001880                 0.400752
```

```
Negative
            Word  review_count  probability  conditional_probability
0   performances           525     0.021038                 0.408415
1    entertaining          555     0.022238                 0.408895
2   unfortunately          323     0.012959                 0.405184
3    particularly          431     0.017279                 0.406911
4  cinematography          374     0.014999                 0.406000
..            ...          ...          ...                      ...
66   deliberately           47     0.001920                 0.400768
67  unintentional          188     0.007559                 0.403024
68   consistently           42     0.001720                 0.400688
69  revolutionary           37     0.001520                 0.400608
70   specifically           40     0.001640                 0.400656
```

Based on these results we can see that the conditional probabilities, that a word is in a review, given the review has a specific sentiment, are approximately twice the probabilities of a word being in any given review. This makes sense given the 50/50 split of the priors.

Task 5

In task 5 I run a Bayesian classifier algorithm, by calculating the log likelihood of a word being in a review based on sentiment. I use Math.log to calculate the log value from the prior, then for each review, I add the log likelihood of the word being in the review.

This value is calculated for both positive and negative values, and the two are compared. Whichever value is larger is used as the prediction.

Once again, I implement a cache, this time saving the predictions returned by the classifier. The amount of time saved with this cache is very large, as the classifier loops are very expensive to process.

Task 6

By the nature of task 6, we have a lot of looping over previous work. The caching implemented in previous steps is hugely important in not repeating processing steps

Sadly, even with the caching, I was unable to complete the full run in time and see the results of Task 6. Instead, I will explain my expectations.

Task 6 loops through the entire project for values of minimum word length in the range of 1-10.

Using several arrays, I store the values of each loop, so I can refer to them by index.

This task also introduces the k-fold step, rather than splitting by value of the "split column". I use the library scikit learn, to run this process using their built-in function. This splits the data into 5 "folds", which are then further broken down into test and train data frames, as they are run through the functions of the previous tasks.

Each fold is trained and used to generate predictions, both in test and training data. Alongside the 10 iterations of minimum word size, this comes to 50 pairs of test and training data, at 100 total models. This is excessive, and I should have cut down the number of folds, and only ran training models, rather than also doing so for testing data. If I had done so, I may have had time to complete the run.

```
print(f"Average Training Data Accuracy: {sum(train_score_array) / len(train_score_array):.2f}")
print(f"Maximum Training Data Accuracy: {max(train_score_array):.2f}")
print(f"Maximum Training Data Score by Minimum Word Length: {train_score_array.index(max(train_score_array)) + 1}")

print_divider("-")

print(f"Average Test Data Accuracy: {sum(test_score_array) / len(test_score_array):.2f}")
print(f"Maximum Test Data Accuracy: {max(test_score_array):.2f}")
print(f"Maximum Test Data Score by Minimum Word Length: {test_score_array.index(max(test_score_array)) + 1}")

print_divider("-")
# print confusion matrix
print(f"Training Data Confusion Matrix")
for i in range(0, len(test_true_pos_array)):
    print(f"Minimum Word Length: {i + 1}")
    print(f"True Positive: {test_true_pos_array[i]}")
    print(f"True Negative: {test_true_neg_array[i]}")
    print(f"False Positive: {test_false_pos_array[i]}")
    print(f"False Negative: {test_false_neg_array[i]}")
    print_divider("-")

    # Percentage of True Positives, True Negatives, False Positives and False Negatives
    print(f"True Positive Percentage: {test_true_pos_array[i] / (test_true_pos_array[i] + test_false_pos_array[i]) * 100:.2f}%")
    print(f"True Negative Percentage: {test_true_neg_array[i] / (test_true_neg_array[i] + test_false_neg_array[i]) * 100:.2f}%")
    print(f"False Positive Percentage: {test_false_pos_array[i] / (test_true_pos_array[i] + test_false_pos_array[i]) * 100:.2f}%")
    print(f"False Negative Percentage: {test_false_neg_array[i] / (test_true_neg_array[i] + test_false_neg_array[i]) * 100:.2f}%")
    print_divider("-")
```

The code above prints several key pieces of information:

The training score for each iteration, which is calculated by adding, the number of reviews which are correctly predicted, divided by the total reviews, for each fold. This is calculated for each iteration of minimum word size.

I then print the confusion matrix for each iteration. Showing the number of correctly classified positives and negatives, alongside those which were incorrectly classified.

Lastly, I output the percentage for correct classifications by sentiment, and the percentages of incorrect classifications.

In hindsight, I believe I should have implemented multi-thread processing and investigate GPU acceleration as an option. This will be considered from an early stage going into the next project.

Aside

I implemented some output formatting functions, to give visual indicators of progress, and make it clear which section of the code was being worked on.

A progress bar was added to areas with expensive loops, to show how far along the process is. An estimate of time remaining is calculated by recording the starting time, and interpolating time elapsed to the value it would be at 100%.

```python
# Function to display a progress bar so the user knows the program is still running and how far along it is
def progressbar(i, upper_range, start_time):  2 usages  Emmett
    # Calculate the percentage of completion
    percentage = (i / (upper_range - 1)) * 100
    # Calculate the number of '█' characters to display
    num_blocks = int(percentage)
    # Calculate elapsed time
    elapsed_time = time.time() - start_time
    # Estimate remaining time
    if percentage > 0:
        estimated_total_time = elapsed_time / (percentage / 100)
        remaining_time = estimated_total_time - elapsed_time
    else:
        remaining_time = 0
    # Create the progress bar string
    progress_string = f'\r{("█" * num_blocks)}{("_" * (100 - num_blocks))} {percentage:.2f}% | Elapsed: {elapsed_time:.2f}s | Remaining: {remaining_time:.2f}s'
    if i == upper_range - 1:
        print(progress_string)
    else:
        print(progress_string, end='', flush=True)
```

```
███████_____________________________________________________________
7.61% | Elapsed: 38.18s | Remaining: 463.30s
```