Assignment 2

Due: Oct 14th (Friday) before 11:59 PM

1. Learning Goals

Learn how to

1. use associative containers like std::unordered_set and std::unordered_map
2. use iterators to iterate through associative containers
3. use algorithms that are available in the C++ Standard Library
4. pass command line arguments to a C++ program
5. partition a C++ program in header files (.hpp) and source files (.cpp)
6. read, understand, and use code written by other people

2. Description

In this assignment you'll be doing the following:

1. Read a raw text file containing movie reviews and ratings.
2. Remove the punctuations and stopwords (e.g. is, and, the, etc.) from the reviews and write the contents to a clean review file.
3. Associate a value for all words found in the clean reviews based on the average rating of the word.
4. Using the map between the words and their associated values, rate reviews from another file that doesn't have any rating already.
5. Write the ratings of the reviews to an output file.

2.1. Clean the data

The raw data with the reviews and the rating will be present in a file named rawReviewRatings.txt. Every review is given a rating between 0 - 4 (both inclusive). The following are interpretation of the rating values:

- 0 = Negative
- 1 = Somewhat Negative
- 2 = Neutral
- 3 = Somewhat Positive
- 4 = Positive

Each line of review follows the following format:
<rating> <review>

Example:
4 The Jungle Book is awesome!

where 4 is the rating and the text “The Jungle Book is awesome!” is the review.
Now, your first task is to read the contents of this file and **clean the data!**

The following steps needs to be performed in the same order for cleaning the data:

1. Read the raw reviews from an input file. e.g. rawReviewRatings.txt

2. Replace the hyphens in every line of text with spaces.
   e.g. If the review contains the word “awe-inspiring”, then it should be split into 2 words namely “awe” and “inspiring”

3. Split each line of text into multiple words using the space character as the delimiter.
   e.g. “The Jungle Book is awesome!” becomes | “The” | “Jungle” | “Book” | “is” | “awesome!” |

4. Remove the punctuation marks from the words.
   e.g. “awesome!” becomes “awesome” and “!” becomes “”

5. If there are any trailing/leading whitespaces, then remove them. Remember that the space character (‘ ’), tab (‘\t’), newline (‘\n’), etc. are considered as whitespaces. For a complete list of whitespace characters this.
   e.g. “zootopia ” becomes “zootopia”, “hello\t” becomes “hello”, and “world\n” becomes “world”

6. Remove the empty words. i.e. words with length == 0.

7. Remove single lettered words. i.e. words with length == 1.

8. Remove stopwords. e.g. is, and, the, etc. The list of stopwords can be found in the file stopwords.txt

9. Write the contents of the clean data to a file named cleanReviewRatings.txt.

2.2. Fill the dictionary

Using the cleanReviewRatings.txt file, create a map/dictionary for every word that is found in this file. The dictionary is of type `std::unordered_map<string, std::pair<long, long>>`. We associate a pair of values for each word namely its total_rating and its total_count. For example, the word “fantastic” has occurred 3 times in the file cleanReviewRatings.txt with associated ratings of 3, 4, and 3 (see the last 3 lines in the file). Therefore the word “fantastic” gets a total_rating of 10 (i.e. 3 + 4 + 3 = 10) and a total_count of 3 since it had occurred 3 times in the file. In a similar way, the other entries in this map are filled up as shown below. You may want to read about a std::pair to understand more about how this map is organized.

word	total_rating	total_count
fantastic	10	3
Zootopia	3	1
Dory	1	1
Finding	1	1
worst	0	1
The	6	2
inspiring	2	1
Jungle	8	2
Book	8	2
good	1	1
Jack	0	1
awesome	4	1
Lion	5	2
King	5	2
awe	2	1
Jill	0	1

2.3. Rate new reviews

Using the map/dictionary that we created in the previous step, we are going to read new unrated reviews from a file (e.g. rawReviews.txt) and predict a rating for each review in this file. The predicted rating for each review is written to an output file named ratings.txt.

How do we predict the ratings for the unrated reviews?

1. Read the input file with the unrated raw reviews (e.g. rawReviews.txt).
2. Clean the data and produce an output file named cleanReviews.txt. The process for cleaning the data is exactly the same as we did in step 2.1.
3. Rate each review by finding the rating for each word from the map/dictionary that we created in step 2.2. The rating for a line of review is the average value of the rating of all the words in the review. If some word in this unrated review is not found in the map/dictionary, then that word is given a neutral rating of 2. If a review is empty (i.e. the review contains no words in it), then such a review is also given a neutral rating of 2.

   e.g. Let see how we computed the rating for the 2nd review in cleanReviews.txt (i.e. “The Lion King fantastic”). We lookup the map/dictionary that we created before and get the average rating for each word in this review. The average ratings of each word in this review is shown below:

The: 6/2 = 3
Lion: 5/2 = 2.5
King: 5/2 = 2.5
fantastic: 10/3 = 3.33

Based on these individual values, this line gets and average review of 2.83 as shown in the file ratings.txt.

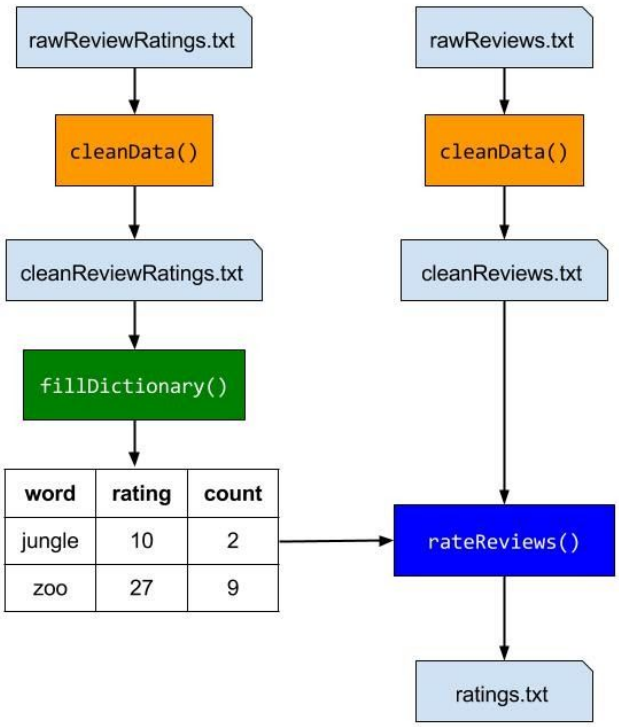
```
Rating for this line = (3 + 2.5 + 2.5 + 3.33) / 4 = 2.83
```
We are dividing by 4 since this review contains 4 words in total.

Another example: “Finding Nemo great”
Finding: 1
Nemo: 2 (because it is NOT found in the map)
great: 2 (because it is also NOT found in the map)

```
Rating for this line = (1 + 2 + 2) / 3 = 1.67
```

4. Write the ratings for all the reviews to an output file named ratings.txt.

A high-level process diagram of the steps explained in 2.1 - 2.3 is shown. Hope this visualization helps you to understand the idea in a better way! :)



That's it! Now, after completing this step you may pat yourself on the back for doing an amazing job with this assignment! :) Also, if you have not taken Artificial Intelligence or Machine Learning before then you have just now completed your first exercise in Machine Learning. You just created a program that may predict the ratings of movie reviews based on some learning it did before based on some reviews that already had ratings associated with them. Well, even though our algorithm is very simple, this is the basic idea behind Machine Learning. Newcomers, welcome to the world of Artificial Intelligence! :)

3. Sample Output

If your program runs correctly without any errors, then NO OUTPUT will be written to the console. If any input/output file cannot be opened, only then your program writes some error messages to the console. All error handling is already taken care of in the file main.cpp.

3.1. Small Data

INPUT FILES:
Rated Review File: rawReviewRatings.txt
Unrated Review File: rawReviews.txt
Stopwords File: stopwords.txt

OUTPUT FILES:
Cleaned Rated Review File: cleanReviewRatings.txt
Cleaned Unrated Review File: cleanReviews.txt
Ratings File: ratings.txt

3.2. "Big" Data

INPUT FILES:
Rated Review File: rawReviewRatingsBig.txt
Unrated Review File: rawReviewsBig.txt
Stopwords File: stopwords.txt

OUTPUT FILES:
Cleaned Rated Review File: cleanReviewRatings.txt
Cleaned Unrated Review File: cleanReviews.txt
Ratings File: ratings.txt

4. Important Details

1. The code for this assignment is split across 5 files.
    a. main.cpp

b. a2.hpp
c. a2.cpp (This is the ONLY file you should modify!)
d. trim.hpp
e. trim.cpp

The .hpp files are the header files which contain only the declaration of the functions and the .cpp files are the source files which contain the implementation of these functions. If you want to understand more about why we have separate header and source files, then maybe you should read this.

2. Take some time to read the source code starting from the main() function that is present inside the file main.cpp.

3. All the code you write for this assignment will only be inside the file **a2.cpp.** You should not perform any file IO, print to stdout, or print to stderr in a2.cpp.

4. The files trim.hpp and trim.cpp are provided only to help you with trimming the whitespaces in strings since doing so in C++ is not so easy as it's in Java. You may read these files to find out how to trim strings for this assignment.

5. You should use the following command to **compile** your program.

```
$ g++ *.cpp -std=c++11
```

where *.cpp means all the .cpp files are to be compiled. You need not mention the names of the .hpp files since they are included from within the .cpp files. The option "-std=c++11" informs the compiler that our may contain C++11 features (e.g. std::unordered_map was introduced only in C++11).

6. Your program takes the two files rawReviewRatings.txt and rawReviews.txt as command line arguments**.** You should run your program as shown below:

```
$ ./a.out rawReviewRatings.txt rawReviews.txt
```

If you don't give these 2 command line arguments to your program, then the following error message will be printed. This error handling is already handled for you in main.cpp.

```
$ ./a.out
USAGE: ./a.out <ReviewRatingsFile> <ReviewsFile>
```

7. If your program works correctly without any errors, then nothing is printed to the console. Only the three output files cleanReviewRatings.txt, cleanReviews.txt and ratings.txt will be produced. You may even try compiling and running your program before adding any

code in a2.cpp to see if these 3 output files are produced. The contents of these output files will be empty until the methods in a2.cpp are implemented.

8. You should NOT add any new functions/methods in a2.cpp.

9. You **SHOULD NOT MODIFY** any of the following in a2.cpp:
    a. Names of the functions
    b. Names and types of the parameters to a function.
    c. Return types of the functions.

10. An approximate number of lines of code is given for each function in a2.cpp. These are the number of lines for each function in Gerald's implementation of a2.cpp. We have provided this information since it may help you to see how little code you may write if you make use of the C++ Standard Library effectively. This would help you to stop and think for a while before you may write a lot of code by yourself and maybe later realize that you could have used some functionality from the standard library to do the same thing with little code and more effectively!

11. Updates to this specification that may be posted on piazza are also part of this specification.

5. Grading Scheme

Item	Grade
Correctness of your code	90
Code Style (indentation) & Implementation Comments in a2.cpp	10
TOTAL	**100**

Hope you have a great time learning and coding associative containers and algorithms using the C++ Standard Library! :)

6. Acknowledgements

Our assignment is based on this nifty assignment by Eric D. Manley and Timothy M. Urness.