**Homework #1 Report on Movie Reviews Sentiments**

**Name:** Hammad Hanif

**Name on Miner:** miner

**Accuracy Rate on Miner:** 81%

**Rank on Miner:** 15th

This project required a great deal of research. The start of this project was not one of the easiest tasks because I never got exposed to a project where data processing was looked-for. However, I started my research by reading the book. It led me to a completely different path, the book explained how KNN algorithm works, it did not help me as far as starting the project was concerned. Resulting, I started looking up some online resources. It helped me somewhat and I started my project by loading the train and test data and putting each line into an array. At that time, I did know what to do with array of all reviews. However, as I progressed, the project started to make more sense.

After loading the data from file into arrays. I remembered from the lecture something about removing the stop words and the punctuations from our data. Hence, I started my research on doing exactly that. After doing several hours of exploration, I found out that the best way to remove stop words by using the library called NLTK (Natural Language Toolkit). The reason it is the best library for preprocessing the data is because it has the best collection of stop words, it also allows a feature to load extra stop words into it. Furthermore, I had to read the documentation on how to use it in my project. In the process of my research, I also got exposed to something called stemmers and a way to tokenize the entire review into words. Then, I read more about tokenize the words and found out that it only splits the word by space, which does not help removing the punctuation. Therefore, I came up with regular expression of my own. My regular expression did not just split words by space, but it also split words by “-”. By doing exactly, I was able to remove some extra words which were unnecessary for my KNN classifier. I did additional research on stemmer and found out that there are many different several different stemmers out there. They all did the same thing, but then again some of them were slightly faster than others. I used stemmer to normalize my data because some of the words in the reviews were not complete or were spelled wrong. Therefore, my research took me to a Lancaster stemmer by NLTK. Note: I used Porter Stemmer in my first implementation, but it did not make much difference. Overall, it took me nearly a day to complete only the pre-processing of the data.

Next step was to vectorize my data and I started my research and it led me to a library called Sklearn. I tried two different ways to vectorizing my data. First, I used countVectorizer to vectorize my data, which did not turn out to be effective for me because all it did was that it counted how many times one word appeared across all the reviews. Then I discovered TfidfVectorizer which converts raw data into matrix form by using TF-IDF features. This part did not take much longer because there was a great deal of documentation available on Sckit website. Following the vectorization, I had to use some sort of calculation method. Therefore, I had two options to do so. First option was to use Euclidean distance, which I used for my first implementation. However, it turned out to be extremely slow and acquired up a lot of my memory space. Therefore, it resulted into memory error. It took me awhile to figure out what was taking that much of my memory space. Suddenly, I looked up the alternative for the Euclidean distance and bump into something called Cosine Similarity. I changed my implementation from Euclidean distance to a Cosine Similarity. It made a significant difference in my run time and I saw my processing got faster.

After implementing cosine similarity, next step was to think of an algorithm to implement KNN successfully. Before I started writing code for KNN, I had to learn how exactly KNN works. For that reason, I started reading the book which helped me a lot. It did not take me long to understand and come up with my first implementation of KNN.

I implemented KNN three different ways. My first implementation had a KNN runtime of O(N^3) which turned out to be very slow. The reason was because I was passing all the test data and all the train data to the cosine similarity. That gave me a 25,000 x 25,000 array as a result. To compare each test review with entire train dataset, I had to use two for loops and one loop to run until k. The entire implementation took almost 5 hours to run.

My second KNN implantation was slightly better because this time my runtime was O(n^2). This time I was only passing 1 test review and the entire train data to the cosine similarity function. Which required only one for loop. Since I already knew that the array was 0 x 25,000. This implementation took utmost 2 to 3 hours depending on K value.

I kept on working to improve my KNN algorithm. After several hours, I was able to find a way to make my algorithm extremely fast. What made it better was the sorting method that I used. Instead of sorting the values in the array, I was able to sort the indexes of the highest values. Which took away the complications that I was having in terms of comparing 2 values with each other. It saved me a lot of memory storage as well as my code started running way faster than before. This implementation took about 30-45 minutes to run for any K value. Below is a table of my testing results. Besides the sorting in the algorithm, the runtime of my KNN algorithm is O(N).

|  |  |
| --- | --- |
| K | % |
| 13 | 66 |
| 19 | 69 |
| 29 | 71 |
| 43 | 73 |
| 99 | 75 |
| 149 | 78 |
| 249 | 80 |
| 299 | 81 |
| 399 | 81 |

Above table is my analysis of how my accuracy increased. I observed that I increased my K and my accuracy got better. However, at some point it stopped increasing no matter how much I increase my K. The given accuracy metrics was one of the most important things to successfully test my algorithm. The fact that test data had 50 percent positive reviews and 50 percent negative reviews. It prevents a person to just guess as all positives or all negatives. Accuracy metrics would have been unsuccessful if the number of negatives were to be 80 percent and number of positive were to be 20 percent. By predicting all negatives would have ended up into 80% overall accuracy rate, which is not a goal for this project.

In retrospect, I can say that I have gained enormous amount of knowledge from this project. However, I could have done better if I knew that this project is not about writing code, but it is more about doing research in a precise manner. I will try to follow the pattern that I follow for this project regards to the researching throughout this semester for this class.