HW 1 – Amazon Review Classification

Introduction:

“A practical application in e-commerce applications is to infer sentiment (or polarity) from free form review text submitted for range of products. For the purposes of this assignment you have to implement a k-Nearest Neighbor Classifier to predict the sentiment for 18506 reviews for baby products provided in the test file (test.data). Positive sentiment is represented by a review rating and given by +1 and Negative Sentiment is represented by a review rating of -1. In test.dat you are only provided the reviews but no ground truth rating which will be used for comparing your predictions. Training data consists of 18506 reviews as well and exists in file train\_file.dat. Each row begins with the sentiment score followed with a text of the rating” (Project Specs).

Approach:

I decided to do this project in Java since I had no prior experience with Python. Using Python may have been easier because of the various built in libraries that assists with many aspects of this project.

1. Preprocessing

First, we need to open both files and do preprocessing to get rid of words or characters that have no significance. We remove all instances of symbols such as punctuation(.). Next, we remove numbers and remove all words that are three characters or less. Then we convert all our words to lower case. Lastly, we have a list of stop words (the, this) that we remove from our document and clean up any excess whitespace. All the following was done using regular expressions in Java.

2. Matrix

Once we have cleaned up our document, we need to store both training and test data into a matrix. For this, I used a list of lists with each line representing a list and within each line, a list to represent the words. We have also stored the labels from training set in a separate list which will be used later.

3. TF IDF

We create a method to calculate the term frequency. The term frequency is the number of times a given term appears in a document. We compute the inverse document frequency which shows the significance of the term using the formula log (total number of documents divided by number of documents with the given term). We then assign the calculated TF IDF weight for each term and create another matrix to store those values. We do this for both sets of data. Next we align the dimensions between our two sets and decide that the max number of columns or features to be 1000.

4. Cosine Similarity

First, we create a method to calculate the cosine similarity between two vectors. Cosine similarity is a metric that measures how similar two vectors are. The formula used is given as 

5. K nearest Neighbors

We need to calculate the cosine similarity of the test vector to all vectors in the training data set with each document representing one vector. We add the similarity values to a list and keep track of the index. Next we sort the list in descending order. We take the top k values which represents the k documents that are most similar. For this case we select k to be 5. Using the indexes that we stored we access the corresponding labels from training set. If there are more positive labels, we assign +1 to the test result. Otherwise we assign -1. Since our value of k is 5 there won’t be any instance where the number of positive and negative labels are the same. We will need to repeat this process for all test vectors.

6. Results

The username registered on miner website is: (kgu2). At the time of this report, the current rank is: 86/147 and the current accuracy is: 58.

7. Instructions

The test file needs to be labeled ‘test.dat’. The training file needs to be labeled ‘train\_file.dat’. The number of lines needs to be 18506. The program will take these two files and generate an output file with +1 or -1 depending on the sentiment.

8. Efficiency

The algorithm itself is not very efficient. For large data sets or in this case, 18,000+ documents, the total runtime could take multiple hours. The total runtime for this case is about 2 hours. My main goal of this project was to correctly implement the algorithm rather than focus on efficiency since that was not taken in consideration for part of the grading. I did not consider multiple approaches to improve run time. Some things that could have been done to improve runtime are feature selection techniques and simply reducing the total number of loops.