Homework 1 Report

By Marcus Domingo

Team Name: anonymousMiner

Current Rank & Accuracy: 10th with 0.81

My Approaches:

I started this assignment the day after it was posted, so before any of the lectures on K-Nearest Neighbor. I have no history of using Python and didn't feel the need to learn it for this assignment because I am proficient in Java. My initial approach was to separate the training data into two sets, negative reviews and positive reviews. For every test review that was read in I would do a binary analysis if the word occurred in the test review and whatever review I was comparing it to at the time. The test review would end up with a positive and a negative score and I would assign it the value based on which the test review favored. This got me to an accuracy of 0.69 on the leaderboard and I was ranked 1st. After a couple of google searches I discovered that for a more accurate analysis you should use term frequency inverse document frequency (tf-idf) as the weight of the word instead of binary. The equation looks like this:

$$a_{ij} = f_{ij} * \log_2 \frac{N}{n_i}$$

In the equation a_{ij} is the weight of the word i in document j, f_{ij} is the frequency of the word i in document j, N is the number of documents in the collection, and n_i is the number of occurrences of the word i in the collection. I was misusing this equation where I was calculating f_{ij} as the frequency of the word i in all the documents and not a specific document. Although my implementation was wrong this produced me an accuracy of 0.80 on the leaderboard and I was ranked 3rd because some other people had started hitting high accuracies.

I decided to try another approach where I would plot the reviews on a "graph" and use distance to find the closest neighbors. I created x (positive) and y (negative) values for all the training reviews and used then find the distance between these points and new test review x and y values. I soon ditched this idea after the matrix lecture and decided to try the matrix Euclidean distance:

$$dist = \sqrt{\sum_{k=1}^{n} (p_k - q_k)^2}$$

I successfully designed and coded the matrix Euclidean distance with and without tf-idf. I continuously tried binary, frequency, tf-idf, and frequency $*$ tf-idf, but none of these would get me above .70 accuracy. I try multiple stop word filters from 5 to 500 stop words and many different k values, but still nothing is giving me that higher accuracy I was looking for with a correct method towards K-Nearest Neighbor.

The assignment due date was around the corner and I didn't want to try another method, I wanted to get this Euclidean correct and I even had thoughts of trying to switch to Python even though I have no experience because I around that Python has good resources. Anyways I decided to stick to Java. I suddenly realized that the implementation I was going for before with the x and y values is just a play on the Euclidean distance since the distance formula for points on a graph is the same, it's just a 2-by-2 matrix. I calculate the rank of a word by adding 1 every time the word appeared in a positive review subtracting 1 every time the word appeared in a negative review. Based on the rank of the word is where I added the weight of the word per review, either x or y. For my weights I had binary, frequency, tf-idf, and frequency $*$ tf-idf. I tested each weight with different sets of stop words and different k values and found that tf-idf worked the best. My k values ranged from 3 to 2001 and I ranged between a 0.78 and a 0.81

depending upon which stop word filter I used. Since tf-idf is supposed to eliminate the use of stop word filters my best results came when I didn't use a stop word filter.

On the final run of my project, where I sit as of now, I used no filters, $k = 201$, tf-idf weight, and Euclidean distance on ordered pairs and received a score of 0.81 on the leaderboard and now ranked 10th.

Methodology:

Euclidean distance seemed to be the most straight forward approach for the data we have. You only have two scenarios, positive or negative, therefore our layout is distinct between where the positives and negatives lie minus the noise reviews. Therefore, if our k is big enough we can ignore the noise and focus strictly on the closest distances from either positive or negative reviews. Using a rank system to rank each specific word was important because it helps with the noise reviews seeing that some positive reviews had a lot of negative words and a lot of negative reviews had a lot of positive words. Lastly the choice to use tf-idf eliminated the use of a stop word filter as well as increased the accuracy of our implementation.

Metric Accuracy:

We have a 2-class problem, positive or negative, and with the training data the positive and negative reviews are evenly distributed among the 18506 reviews. There are exactly 10000 positive reviews and 8506 negative reviews. The accuracy metric for both are:

$$positive = \frac{10000}{18506} = 54\%, negative = \frac{8506}{18506} = 46\%$$

Therefore, the accuracy metric is suitable for our case because about half the time positive reviews are detected and the other half negative reviews are detected. A case where this would be unsuitable is if the training data was lopsided like the example given in the PowerPoint *W2_data_classification* slide 71 because then only one of the classes get detected while the other one doesn't.