HW 1 Report

Username on miner is psubedi4. My current rank is 357 and score of .54 on the miner website. You can run my program as is, currently it has a test case. It is using a file named train\_file\_test.dat to substitute for train\_file.dat. This file contains 202 reviews to encapsulate the population. Different number of reviews were used for testing, but this is currently the number of reviews in the train\_file\_test.dat. I also created another new file test\_test.dat which contains 1 review. This is the prediction file and the 1 review will be plotted on the “graph” that KNN creates and then this review will be predicted. You can run this program with the correctly named files if you change both string in the with open statement in python on lines 33 and 68.

Currently if you run the program it will run on the files train\_file\_test.dat and test\_test.dat they are the training data and test data respectively. Change lines 33 and 68 to the name of the tiles you want to run the program with. First file opened should be the training file and the second should be the test file.

this was programmed in the IDE PyCharm. I was a fan of intellij so it was an easy switch to PyCharm, both developed under the same company. I am using the packages nltk, string, item getter from operator, scipy, and textblob.

I used python, no packages, to first open the files. The training file was the first one to open because I had to create a model for my data, so that I could make predictions. The data was given so that each review was stored in one line of the file. After opening the file, I read through every line also every review one by one. The first thing that you need to do is clean your data. Your data is messy before the cleaning and will not be too helpful in the state that it is in. In my approach I first logged if the review was positive or negative. You could tell whether a review was positive or negative by the plus one or negative one as the first two characters in the line. This was stored into an array for future reference. This decision was made because later when cleaning up the data the plus or minus would be gone, and we would have no reference back to whether the review was positive or negative. This would be a problem if we ever wanted to debug. After saving the status of the review I would tokenize the sentence so that each word would be separable from the sentence. No fancy library was used simple python split function to get the tokenization working. Lower cased all the words and removed all punctuation. Stemmed the words using nltk porter stemmer. Stemming the words was done so that there would be more a uniform set of words to swift through. Stop words were removed next also using nltk. Removal of stop words was done because we only need the critical or useful words and other words such as the or is are meaningless to our algorithm.

My implementation includes polarity of each sentence, polarity of the reivew, length of the review, and average polarity of each word. There were also a few features I left out of my final algorithm because I thought they scattered my data points. Some of the features I left out were number of words with polarity over zero, number of words with polarity under zero, percentage of good words, percentage of bad words, and number of nouns. I chose to leave out number of nouns because it made my programs runtime too long. The average time to finish predicting one review was around four to five seconds which would have taken hours to complete on the real data.

After extracting, cleaning, and getting my features from all of my training and test set I start working on my KNN algorithm. My implementation of KNN used arrays to store the test set and the training set. I would find the Euclidian distance between the test set point and the training set points that are already plotted. I chose to use the Euclidian distance because length of the reivew was not an issue for me. If I were to use wordcount as a feature I might have used cos to normalize my data. I chose five as my K because it seemed to work best when testing my data. Finally, my program writes to a file that will print plus one for a positive review or negative one for a negative review.

There were many opportunities that presented itself where I could speed up run time. In the training file I first went through each word on the line more than n squared times because I was going through them each time for my features. I cleaned this up to where I am doing three runs which is N cubed time for the features however, this does consider t the huge overhead of going through each line of the training data and each word of each line.

I implemented cross validation in two ways. The first way was described above where I created two new files for the training data and test data. This was because it was simple and fast move a small amount of different reviews into these files then test. The second method I used for cross validation was splitting the entire training data into an eighty twenty split. Eighty percent of the training data would be used to model the data while the other twenty percent was used as new data point (reviews) that had to be predicted.