**Shane Armstrong**

**CS 584 Team Name - H@ck3rm@n**

**Homework 1 – Amazon Review Classification**

1. H@ck3rm@n
2. Rank – 15; Accuracy - 0.74

# Approach

I broke up the project into stages. The first stage was getting aquited with the data. I looked through the reviews to get an idea of what the data looked like and what precautions I should take while reading in the data. Discoveries included: empty lines, misspelled words, hyphenated words, and unorganized punctuation. I have not used Python in years so the next stage was refamiliarizing myself with different available built-in methods and relevant modules. I first looked online for different Python3 tutorials and became acquainted with Python and different available IDEs for Windows and different configuration options for VIM on Ubuntu. The next stage was learning how to clean the data. I spent a lot of time learning what needed to be done to clean text data for sentiment analysis. This search led me to different packages available for use with Python. The most commonly cited packages online seemed to be NLTK and scikit-learn. From there I experimented with different capabilities and decided to focus on the use of NLTK to clean the text data. NLTK and scikit-learn had many packages that would have made this project very simple possibly even trivial, such as its own kNN method, dictionary creator, data validator method, etc. I elected to just use NLTK tokenizer and stop words corpus to avoid being overly reliant on the packages and accidently not code important parts of the projected expected to be performed by us.

For the data processing stage I used NLTK tokenizer. I originally performed the data processing in the same program as the classification program. After I implemented a working solution, I realized that I would need to make the kNN method as lightweight as possible. I processed the reviews removing common words and some punction, then storing the new file as a CSV. I performed similar actions with the dictionary created from my reviews and the SentiWordNet. For the dictionary I created, I looked at all training reviews and stored each every word in a dictionary. I tracked the frequency and the net sentiment of each word (+1 if the word was in a positive review, -1 if the word was in a negative review). I removed the 70% least used words and words that had neutral net sentiment. I stored my frequency dictionary and sentiment dictionary as JSON files to be able to easily review the dictionaries by hand and to quickly read them into my kNN program. The SentiWordNet had to be cleaned to be used for my purposes. I choose based on reviews of their software project and that they freely distribute for use to modify and use, even commercially. I stripped and cleaned the document and converted it to a JSON object as well.

The next stage was the cross data validation stage. I developed the cross validation program to break up training set into different partitions and use those partitions to predict the classification of training reviews based off all available training review data. For instance I did not recreate the training dictionary from each training partition. I created the training dictionary prior then read it in at the beginning of the run. I used the validation program to test different attributes and how many threads should be used when multithreading. I calculated the error rate of different runs with varied values.

The prediction stage was used to do tweaks to different attributes. The predictions were written to a txt file.

# Methodology

I used normalization, weighted voting based on distance, weighted distance, a sourced sentiment dictionary, a created dictionary, and the length of the review to calculate the k nearest neighbors. The attributes I used were: net sentiment calculated from a sourced dictionary SentiWordNet, net sentiment calculated from a developed dictionary based on frequency and net number of positive and negative reviews, and the length of the reviews. The next attribute I would have implemented would have been a % word similarity. Normalization was necessary because of the magnitude differences between the different attributes. The SentiWordNet sentiment values came normalized for word but I normalized based on the review sentiment. All attributes were assumed to be linear for normalization purposes. Using my data\_validation program I manually tested different variables, it did not optimize on a given range of a given variable. This would have been my next update to the file. Currently it loops through a certain range and prints the accuracy and attribute value to a text file to be manually read. The partition allocation was generalized, so a given number of partitions were given and it would loop by testing each partition. The generalization could have been skipped and just developed to do a test partition of 1 with all reviews as training. I found that k was best at approximately 30. The created dictionary was prone to overfitting and the sourced sentiment dictionary was better at consistently predicting sentiment. I found a slight relation to the length of a review and its sentiment but it could be left off (I left it in my program with a very small weight).

# Conclusion

I enjoyed this project and I wish I had more time to tinker with my attributes and make the code run more efficiently. I incorporated multithreading but I ran into issues while trying to implement multiple processes. There seemed to be an issue with scope when normalizing. It took approximately 15 minutes to run with multithreading. On the next project I will try to incorporate more efficient data structures to decrease the time to run.