Ibrahim Munir
CS 484-001

# HW1 – Movie Review Classification Report

Miner user/GMU User ID: imunir2/imunir2 Rank: 60 Accuracy: 0.81

## *Preprocessing the Data*

In order to use TF-IDF on the data (discussed later), the data needed to be preprocessed to maximize the potential of the classifier and lead to accurate predictions. To do this, parts of the reviews which seemed to not add any value or sentiment were removed. HTML tags, punctuations, stop words, compound adjectives, and unnecessary symbols within the text were extracted to produce a cleaner version of the text. All letters within the review were converted to lowercase to ensure uniformity among the words (e.g. so words like 'READ' and 'read' would be interpreted the same). Lastly, all words were stemmed to their bases to continue to ensure uniformity (so words like 'watching' and 'watched' were stemmed to 'watch', thus interpreted the same when using TF-IDF).

## *Engineering Features from the Data*

Even with the data preprocessed and organized in a clean manner, the data still had to be manipulated to a format in which the classifier could understand clearly. The technique used for this purpose was term frequency-inverse document frequency (TF-IDF). TF-IDF would allow me to translate the preprocessed data into a numerical form, specifically vectors that had n dimensions, n being the size of the corpus. Each word would have its own dimension, and each review would have a number in each dimension representing how important that word was in the review. Such vectors world allow my classifier to easily absorb and make predictions based on the data it had been fed.

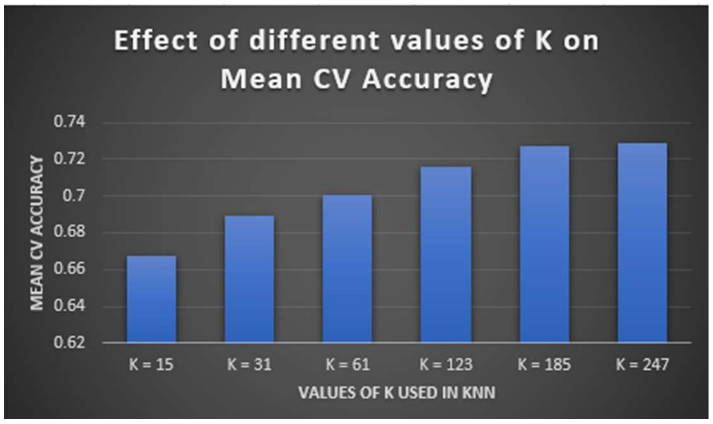
To facilitate this task, a TF-IDF vectorizer was used to translate the preprocessed data into vectors. First the vectorizer was fitted to the training data to learn its vocabulary and inverse document frequency, and then transformed the training data to produce the vectors. For the test data, the vectorizer only transformed the test data, but was not fitted to the test data.

## *Cross-Validation and Hyperparameter Tuning*

In order to build and select the best possible model, hyperparameters for the k-nearest neighbors (KNN) algorithm had to be identified. After long consideration, the value of k, the distance metric, and neighbor voting method were picked as the hyperparameters for my KNN algorithm. To find the most optimal hyperparameters so a robust classifier could be created, I decided to perform k-fold cross-validation. I chose 5 folds as it would offer very good results and be less time expensive.

## *Finding an optimal K*

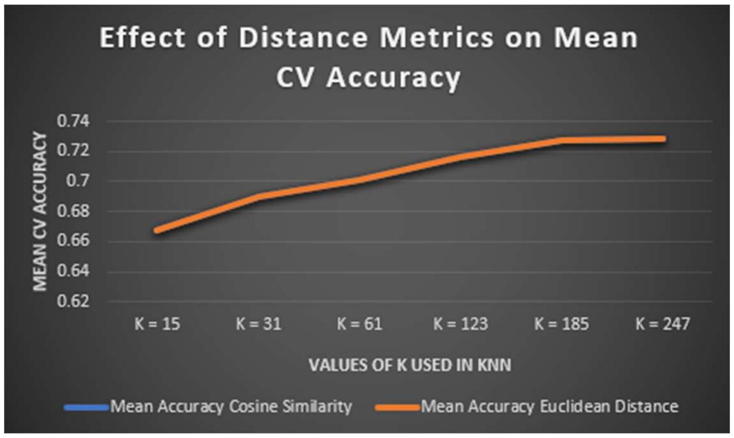
My first main objective in the model selection process was to find an optimal k value to work with in my KNN classifier. After some research, I learned that the general rule of thumb was to start with a k value that what the square root of n, where n denoted the size of your training dataset. Taking the square root of 14,999 (size of training dataset) gave me 122.47. I rounded this number up to 123 since I intended to only use odd values of k to avoid tiebreakers with the two labels (positive or negative) when using the majority voting method (discussed later). I then divided this number by 2 to get 61, divided by 2 again to get 31, and again for 15. 15, 31 and 16 were the values I used when experimenting with k values on the lower end of the spectrum. Since I wanted to see the effect of higher k values as well, I added 62 to 123 to get 185, 62 being the number you get when subtracting 61 from 123. Then I added 62 once more to 185 to get 247, which would be my extremely high k, contrasting the extremely low k value of 15. I then performed 5-fold cross-validation on each of these six k values (15, 31, 61, 123, 185, 247) using my KNN classifier. Note that cosine similarity and the majority neighbor voting method were used as 'default settings' for the other hyperparameters.



As seen in the graph, the higher values of k performed much better when compared to the lower values of k like 15, 31, and 61. I suspected that this was likely because these k values were a lot more susceptible to noise points. A quick observation of the results may have led me to settle upon the highest value 247, however, I feared that picking such a high value for k would sacrifice the ability for the model to generalize well on unseen data points. Additionally, these extremely high k values of 185 and 247 did not outperform the k value of 123 by a huge margin. After deliberating a little bit, I decided to strike a balance and go with a k value of 123 since I did not want to cast such a huge net and include points from other classes when making predictions with very large values of k.

## *Choosing a Distance Metric*

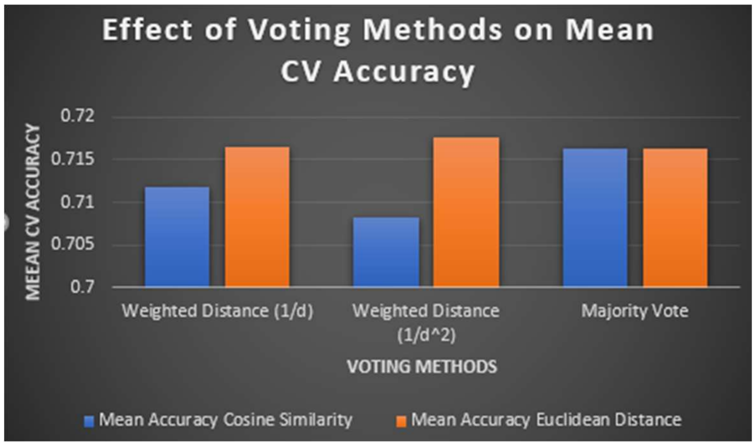
Another hyperparameter I wished to experiment with was the distance metric, namely euclidean distance and cosine similarity. Euclidean distance measures the distance between the two vectors while cosine similarity attempts to measure how close two vectors are pointing in the same direction. To ensure that the accuracy of the k values did not substantially increase depending on the distance metric used, I experimented with the same six k values again. Note that the 'default' majority voting system was used.



Surprisingly enough, distance had no effect at all when I performed 5-fold cross-validation. The mean accuracies were the exact same, so much so that when visualized, the line for euclidean distance covered the line measuring the cosine similarity accuracy on the graph. As a result, I delayed my decision on a distance metric and moved on to testing the next hyperparameter.

## *Deciding on a neighbor voting method*

The last hyperparameter, voting method, was experimented with last. The choices were between a majority and weighted distance method, where I was able to use a weight of $\frac{1}{d}$ or $\frac{1}{d^2}$. I planned to see whether or not the classifier was significantly affected when weighted distances, which emphasized closeness to the test data, were used versus majority voting which stressed the quantity of neighbors over distance to the new data. Since I was still unsure of the best distance metric to use, I tried using these voting methods along with the 2 different distance metrics to see if there were any interesting results. I chose to use k value of 123, which I had settled upon earlier.



Interestingly enough, the $\frac{1}{d}$ weight performed better than the $\frac{1}{d^2}$ weight with cosine similarity whereas the opposite occurred with euclidean distance. However, these were small improvements over each other, and the majority voting system was almost as accurate as the other voting methods.

## *Conclusion*

Ultimately, the only hyperparameter that seemed to have a significant impact on the accuracy of the classifier was k while the distance metrics and voting methods seemed to not have been as impactful. As a result, my final classifier used a k value of 123 as decided earlier while my distance metric and voting method used were reverted back to their 'default' values of cosine similarity and majority neighbor voting since these are the standard hyperparameters widely used when performing KNN.