Joseph Shaffer
Shaffer.567

Data Splitting

I split up the data so 60% of the data is in the training set, while 20% is in the validation set and the other 20% is in the testing set. The data is split up by randomizing the indexes of the sentences and then 60% of the randomized indexes are placed in the training set, 20% is placed in the validation set, and 20% is placed in the test set. This means that every time the program is run, every set will contain different sentences. The program can also be changed so a different percentage of the sentences appears in each different type of set. For example, if you input 30 into the randomized function, 30& of the sentences will be placed in the training set, 30% will be placed in the validation set, and 30% will be placed into testing set.

Feature Selection/Dimension Reduction

I first reduced my feature vector by removing all words that appear in about the same number of sentences with a positive sentiment as the same number of negative sentiment. This is done by seeing if the ratio of the number of positive sentences minus the number of negative sentences for that word, divided by the total of number of sentences that word appears is less than some set threshold. The threshold is set at 0.4. I did this because if a word is basically appearing in the same number of positive sentences as the number of negative sentences it appears in then that word is not a good predicter for the sentiment of a sentence. This is because, if that word appears in a sentence, it is basically equally probable that sentence has a positive sentiment or a negative sentiment.
The second way that I reduced the feature vector is by limiting the feature vector to the K-most frequent words, which I set at 1000, but could be set to any value of K. This was done, because I already have all the words that are a good predicter for a sentence's sentiment, but want to remove all words that do not appear that much, especially if those words appear once. If a word does not appear that often or only appears once, then the sentiment of the sentence that word appears in could be the deciding factor in the prediction of the sentiment for a particular sentence, and I want to utilize the words that occur most frequently and are a greater representation of the data when making a prediction for the sentiment of a sentence.

Classification Algorithms

The first classification algorithm that I used is the K-Nearest Neighbor (KNN) algorithm. I first find the optimal value of K from the options of 1, 3, 5, and 7. This is done by running the validation set against the training set for all different values of K and finding the resulting accuracy of the number of correct predictions over the total number of predictions for each value of K. Thus the K value with the greatest accuracy is what I use when running KNN for every test case. I run the KNN algorithm for the training set, validation set, and testing set against the training set for both the original feature vector and the pruned feature vector to see how the accuracy changes.
The second classification algorithm that I used is the Naïve Bayes Classifier. I implement this algorithm first by finding the probability of a sentence being positive or negative from the number of positive sentences in the training set over the total number of sentences in the training set, and the number of negative sentences in the training set over the total number of sentences in the training set. I then look at each sentence of the set I am looking at, and then at each word of that sentence to find how many positive sentences that word appears in and how many negative sentences that word appears in and divide the total positive sentences for that word by the total number of positive sentences, and then divide the total negative sentences for that word by the total number of negative sentences. I then multiply all these probabilities together for a positive sentiment and negative

sentiment, and the greater probability is what determines the predicter for that sentence's sentiment. I then find out if that prediction is correct and find the accuracy of the classifier by the total number of correct predictions over the total number of predictions.

I found that the Naïve Bayes Classifier is more accurate than the KNN classifier for both the original feature vector and pruned feature vector for the training set, validation set, and test set. However, I found that I see a greater increase in accuracy between the original feature vector and the pruned feature vector for the KNN classifier. Furthermore, I found that the accuracy increases for every set between the original and pruned feature vector, except for the training set in the Naïve Bayes Classifier, where I saw a decrease in accuracy between the original feature vector and pruned feature vector.

Time to Build Model

The time to build the KNN Classifier was about 2 hours and the time to build the Naïve Bayes Classifier was about 3 hours in python. This takes into account the testing to make sure that the classifiers were behaving as they should be.

I also found the time to classify a tuple of the test set in the original feature vector was about 0.004 seconds for both classifiers. For the pruned feature vector, I found that the time to classify a tuple for the KNN classifier was about, 0.0018 seconds and 0.000736 seconds for the pruned feature vector, which means that after pruning, I saw a dramatic increase in the time it takes to classify a tuple with the each classifier. This paired with the fact I saw an increase in accuracy means that the pruned feature vector is faster and a better predicter of the sentiment of a sentence in the test set. The classification time is faster, because the classifiers no longer after to run through every word of every sentence, but can focus on the words that are most important in the classification of the sentence.

Classification Results

For the original feature vector, I found the accuracy of the KNN classifier to be between 60-70% for the training set, testing set, and validation set. Then for the pruned feature vector, I found the accuracy to be between 70%-80% for the training set, validation set, and testing set, which means there was about a 10% increase in accuracy after pruning the feature vector.

For the original feature, I found the accuracy of the Naïve Bayes Classifier to be between 70%-80% for both the validation set and testing set, but found an accuracy of close to 100% for the training set, since it is run against itself. Then I found that with the pruned feature vector with the Naïve Bayes Classifier, the accuracy of the training set, validation set, and testing set was between 80%-90%, which means I saw about a 3% to 7% increase in accuracy for the testing set and validation set, but saw a decrease around 10% in accuracy for the training set. I believe this is due to the fact that by pruning the feature vector I am removing words that could play a greater role in the prediction of the sentiment of a sentence in the training set, thus when running the training set sentences against the training set itself, that pruning might be removing the unique words of a training set sentence that play a bigger role of predicting a sentence's sentiment for the original feature vector. However, with the increase in the accuracy for the test set and validation set, this means that pruning the feature vector is still advantageous when comparing a sentence against the training set in the Naïve Bayes Classifier.

From my results, I can conclude that pruning the feature vector results in not only a faster classification time per tuple but a greater accuracy in classification for both the KNN classifier and Naïve Bayes classifier. Although it is unlikely that someone would run a training set against itself in a classifier, the fact that there is a decrease in the accuracy in the Naïve Bayes Classifier for the training set, means that pruning the feature effects the decisions of the Naïve Bayes Classifier and could make the wrong prediction in some cases.