Spam Classification Using SVM

To get started, we thought back to our own experiences and identified common indicators of spam emails. This was done in an effort to contextualize the assignment. Afterwards, we looked through a few spam and ham email files that were tokenized to identify common patterns and common words, phrases and letter/symbol combinations. In completing the portion of the homework due on Monday we had created a dictionary with the words in both spam and ham emails and sorted it by the greatest discrepancy. This dictionary contained a tuple that had the frequency of ham divided by the total number of ham emails and the other value in the tuple was frequency of spam divided by the total number of spam emails.

Moving forward with the assignment, we changed our approach slightly and only sorted our dictionary based on the frequency of words instead of their proportions. We performed two comparisons using the difference and ratio of each word’s count in ham and spam, and found that the first 150 entries of difference operation and first 76 entries of the ratio operation were particularly indicative of spam vs ham emails. We therefore added both these subsets into a single set (to avoid repeating the same word) and subsequently created an enumerated dictionary where the key was the word and the value was the feature number in the SVM model. This was performed to allow quick lookups and was particularly relevant later on (when we added many more features). We ran our algorithm with the features generated from this list of spam indicators. Out of curiosity we also ran our algorithm using the list of words you (Professor Scharstein) provided. We realized the difference accuracy was quite small. Going off of our data mantra, we decided to combine the two sets of words and our accuracy resulted in about 87%. To increase the number of spam indicators thus increasing the number of testing features for the emails, we iterated through all the words in the emails; with an increase in the number of words (more data), there was an increase in our accuracy. At this point the accuracy was approximately
97%. We were also calculated precision and recall to aid in our interpretation of the effects different things we tried had on our resulting accuracy.

At this point we decided to experiment with parameter C. We randomly tried C=1 for which the accuracy slightly decreased. Then we tried C=0.1 to C=0.9. There was no substantial change in accuracy, precision or recall in comparison to C=1. We looped through C=1 to C=100 and each iteration gave the same output: training - 97.18/96.30 (p/r). We then tried an large value for C (C=10000) and obtain a 100% accuracy on the training set (overfitting) but still 97% on the cross validation set. To see the extreme opposite we tried C = 0.000001 which gave u 55.8% on the training set and 55.25% on the cross-validation set. Decreasing C to 0.01, increased the bias of the svm and decreased the intense overfitting. 0.01 turned out to be the best parameter for our algorithm with a test set accuracy of 98.85% and cross validation accuracy of 98.20% The p/r on the test set was 97.82/99.62 and on the cv 96.84, 99.22. After doing all this experimentation we realized that changing the default c parameter didn’t increase our accuracy by very much.