The first optimization we implemented was the use non-binary features. This allowed us to use frequency rather than occurrence of selected words. Next, we tried using a feature list consisting of all of the unique words in the training data (about 70,000 features). This proved immediately problematic upon implementation because the program took a very, very long time to run. In order to have our program run in 30 minutes, we approximated that we could use about 40,000 features. Following this discussion, we decided to generate a word list that consisted of words that were more likely (by some multiplier) to appear in spam than non-spam emails and vice versa. We then added functionality to extract the most common single, double, and triple spam words as compared to non-spam words and vice versa. Next, we changed how we were generating SVM files such that the generation ran in constant-time. This allowed us to use a much larger feature set (i.e. a feature size that used to take the program 30 minutes to run was now running in less than 30 seconds). Next, we started changing our c-values around. However, at this point we realized that our precision and recall was surprisingly low (hovering around 85%), especially given the huge feature list we were reading in. Embarrassingly, we realized we had spent all of our time running the svm_learn command without ever running svm_classify. Our precision and recall jumped up 10+% immediately when we realized we needed to run svm_classify. *Note to self:
read the homework instructions!* Finally, we developed a c-sweeper that was capable of sweeping back and forth across a range of c-values to find a c-value that provided the best possible accuracy on the given feature list. Before submitting, we trained on all 8000 examples in the training file. Ultimately, our best results were 99.04% accuracy, 98.03% precision, and 99.83% recall.