An initial attempt was made to replicate the svm files (the monday homework) using vectors. This worked but later we found that using dictionaries gave us far more flexibility with future implementations.

We eventually added the function run_svm which allowed for the automated running of the command line prompts, speeding up the time it took to run tests.

We then set out to make our own word list. To do this, we decided to find the 1000 most frequently used words in both the ham sets and the spam sets of the large data set. This resulted in less than 2000 total words/features, as duplicates were removed. By increasing the amount of words we were analyzing for both ham and spam from 1000 each to 2000 each, we were able to raise the accuracy a few tenths of a percent. Increasing the list arbitrarily high from there resulted in no increase in the accuracy of the program on our CV set.

We initially used binary features for the svm, but soon realized that actual frequencies were a much better idea, as they gave better accuracies. However we needed to come up with a way to normalize the frequency counts. We were thinking of normalizing by the maximum word frequency per file, however we came up with a better idea to normalize by what fraction of the email each word is e.g (freq)/(total words). This gave us much better accuracies.

We also thought that it would be a good idea to ignore some of the most frequent words/characters such as “the” “.” and “,”. Which helped at first, until we switched to our final
method of normalizing, where it was actually more accurate when we included these words. This makes sense as an email that is 90% the word “the” is probably spam.

We also considered doing a form of spell checking using PyEnchant, so that one feature could be the percentage of words in a given email that are misspelled, as emails that are mostly misspelled are likely spam. We could have also used this library to correct misspelled words and then count them in the frequencies for that word spelled correctly. However, we found it tedious to install the PyEnchant library, and this was never implemented.