The first thing that we tried was to change the features from binary (1 if it exists) to a count (3 if the word shows up 3 times in the example file). This did not help.

We then tried to change the word list from given word list to the most common words. We quickly realized that we should exclude punctuation. We also then excluded words that were less than 4 characters to remove many common words such as “the” and “I” that would show up in all the emails. From there, we attempted various versions of word list length, finding that 1000 words produced high overfitting and a sweet spot somewhere below 50. We found this to be suspect, however, and eventually realized we needed to split most common words across ham/spam (see below).

We attempted to manually inspect the words and add some that seemed useful but found that adding just three manually selected words to the wordlist dropped our accuracy from 82% to 51% on the test set so we abandoned this strategy.

Just for fun we ran the program using only the word “enron” as our entire list and found that we got 71% accuracy.

After all of these tests, we seemed to hit a roadblock where we were unable to get above ~83%. Finally, we had a breakthrough in the word list creation. We created two lists: one with the most common spam words and a second with the most common ham words. We then took the symmetric difference between these two, which created a list of all the most common words that showed up in only one of the two sets. This allowed us to identify words that were most indicative of one set or the other. Implementing this got us up to 92% on the first run before even adjusting our word list size or C values.

In order to fine-tune our parameters for C and the word list size, we wanted to loop through different values for those parameters and see which ones worked the best. We first tried adjusting the C code in svm_classify in order to return useable output (accuracy) to our loop, but we were unable to get non-integer numbers because of the requirements for main(). We ended up just parsing the output text and isolating the accuracy numbers. The final code includes a large chunk that’s been commented out. This includes the two loops that we used to find an optimal C of 0.2 and optimal wordList size of 1280.