Our main ideas for extracting word lists of features were:

1. Take all words from the first 50 (or whatever number chosen) emails in the data set. This seemed way too simple, but worked very well. Of course it probably contained many words that would never appear again and therefore were not very helpful, but clearly this approach extracted enough features to make predictions with 98% accuracy on our validation set.

2. Given the first 2000 (or whatever number chosen) emails in the data set, for all the spam emails make a dictionary of words and their frequencies, and do the same in a separate dictionary for the ham emails. Then take all the words that occur more than 2 (or whatever threshold chosen) times in one or the other and use those words as our word list. We got similar results with this approach, getting a few tenths of a percent higher than with the first approach, and so we kept this model of feature extraction as our final implementation.

3. In makesvmfile, add the length of the email (number of words) to the end of the svm file as an additional feature (whether the length is greater than or less than some threshold). To determine the threshold, we used the program to calculate the average length of all emails, of spam emails, and of ham emails in the whole data set as well as the min/max values for spam and ham emails. We found that on average, ham emails are longer than spam and that ham emails also had a higher maximum length. So using roughly the longest spam email (4000) as a threshold, this additional feature was added. It did not seem to affect anything, so we got rid of it.

4. We tried using word frequencies instead of the binary version, but the accuracy was lower, so we stuck with binary features.

Additional things we implemented:

1. Faster implementation of makesvmfile using a set to check membership
2. Including subprocess.run() to learn and classify by running the python file directly instead of wasting time at the command line
3. We were unable to figure out how to do a parameter sweep in python, so we used a bash script to run spamsvm.py with a range of c values and extracted the accuracies from each.

4. In order to examine which emails were misclassified, we wrote a python program that opens output.svm and training-labels.txt and compares the labels to the output values to return the numbers of emails that were misclassified. It prints all the misclassified email filenames.

5. Upon examining these misclassified files, we learned nothing but lost faith in whoever made the dataset (there are emails listed as ham that are so clearly spam, e.g. 7503--ham.txt. Perhaps it's worth taking a look at the data for the future). We tried adding all their words to the feature list, expecting to fit the validation set too well, but it did slightly worse than usual (since the training data didn't find them useful?), so we removed them.

Note: We have been tuning our algorithm to get the best accuracy. We've noticed that our recall is higher than precision, meaning that more ham is misclassified than spam. Ideally, for a real spam filter, we would want all ham to go through at the expense of letting some spam in as well.