Below are all the different ways we tried to create feature vectors and the approximate accuracy we obtained. In all the examples, we created our word set by taking the 5000 most frequent words in our training data.

**Binary alone ~ 98 %**
Index all words, then report a 1 or 0 whether or not a word is found in an email.

**Frequency alone ~ high 97 %**
Index all words, then report the number of appearances of a word in an email.

**Correcting all misspelled words then use binary features ~ 90 %**
We preprocessed all the emails in the training set by checking for deliberate misspellings and correcting them, then created svm files with binary features.

**Freq and proportion misspelled ~ high 97 %**
Find #(misspelled) / #(words) for each email and use as an additional feature (additional to the number of appearances of each word in email)

**Binary and proportion misspelled - low-to-mid 98 %**
Find #(misspelled) / #(words) for each email and use as an additional feature (additional to the words that appear in email)

**Words sorted by ratio: (word appearance in spam) / (word appearances not in spam). Then use freq gives ~77% accuracy, however it gets 100% precision (and ~48% recall)**
While finding all words in all emails: store how often they appear in spam, and how often they appear anywhere. Use the ratio #(appearances in spam) / #(appearances in general) as the key to sort the word list. Then use binary features for the top x words.

To find our final hyperparameters, we wrote a parameter-sweeping loop that iterated over values of C. The randomness of our train/test split made it difficult to determine the very best value of C, but C=0.4 seemed to perform the best over multiple train/test splits.