My process began with cleaning the data. After reading in the two text files, I converted them from list type to string type so that I could separate the reviews in each file. I removed all text that came before and after the reviews in each file and converted the reviews in each file from one string to a list of strings by splitting the files by the characters ‘ “,” ‘ that separated each review.

For step 1 of the assignment, using the random and math libraries, I shuffled the reviews in each file and split them into training data (70% of the reviews), development data (15% of the reviews), and test data (15% of the reviews). I also used set.seed to ensure that the data was split in the same group of reviews each time I ran the code. I printed the size of each data set to confirm that they were split correctly. This gave me a training set of 3,731 reviews, and a development set and a test set of 800 reviews each for the negative review data and for the positive review data.

For step 2, I removed all punctuation marks from the training reviews and made all the letters lowercase. This was to ensure the vectorizer did not view “yes!” and “Yes” as different features. I then created a function that takes a list of negative reviews and a list of positive reviews as inputs. It initializes an empty data frame, and creates a column to keep track of the reviews, and another column to store the sentiment of each review (i.e. “pos” or “neg”). It removes all punctuation marks in each review and makes all letters lowercase. Values of all columns are initialized to 0. The count of the occurrences of the unique word replaces the 0 if the unique word is present in the review. It then puts the information into a data frame. This returns a data frame. I ran the negative training data and positive training data through this function. I then filtered to remove features that occurred in fewer than 5 reviews and more than 5,000 reviews. This took the resulting dimensions from (7462, 17299) to (7462, 3339). The index for each unique word (column title) is stored in a dictionary and the data frame is then converted to a numpy array. I used these column titles as my new unique words list and created a new vectorizer function that was the same as the previous one but utilized this smaller list of words.

For step 3, I used the logistic regression scikit-learn classifier. I split the training numpy array into train_y (sentiment) and train_X (values that represent the presence of a word). I then trained the classifier on this data. I then ran the development data through my vectorizer function, split it into dev_y and dev_X, and used the classifier to predict the output. I got an accuracy of 73.25% on my development set. I began adjusting the hyperparameters and noting their effect. Changing penalty to l1 (as well as solver to liblinear) decreased my accuracy to 72.75%. Using the default hyperparameters and only changing C to from 1 to 0.1 resulted in increased accuracy (74.44%). Changing C to 0.01 reduced accuracy to 68.31%. Using 0.1 as C, I tested the effects of adjusting the different hyperparameters and found that a penalty “l2” and solver “lbfgs” along with the other default parameters resulted in the highest accuracy (74.44%). Adjusting the class weight throughout this process did not appear to affect the accuracy.

When I ran the test data through the vectorizer and the best logistic regression model for step 4, I got an accuracy of 75.06%.

For step 5, I used CountVectorizer and TfIdfVectorizer to generate the input vectors. This resulted in a 73.88% accuracy on the development data. After trying out different combinations of hyperparameters, I found that the default hyperparameters resulted in the highest accuracy for the development data.

When using this classifier on the test data, I got accuracy of 77.12%. The classifier created using the CountVectorizer to generate input vectors resulted in a higher accuracy than the manual method. The CountVectorizer method of classification also used removal of stop words. This may be a better method of removing words than the method that I used (removing words outside of a frequency range) and may have contributed to the higher accuracy.