Homework 4

Martin Zugschwert

1) As you can see below, I separated out the training and test data into 80% training and 20% test data by figuring out the size of the “master” file (the file containing all of the reviews) and then sliced the array along the 80% mark and stored that into the train numpy array and stored the other 20% into the test numpy array. This is random each time because I randomly shuffle the values in the “master” numpy array each time the code runs shortly above this point in the code.

A screenshot of a cell phone

Description automatically generated

Here is the output of this code:

A close up of a logo

Description automatically generated

As you can see, there is no duplication of data and the whole data set is now split very evenly into 80% train and 20% test.

2) First thing I do for this step is create a list of all of the words that occur more than 4700 times or less than 10, seeing as these words are not relevant / would massively slow down my program and create numpy arrays that are much too large for my computer’s RAM. Code to do this:

A screenshot of a cell phone

Description automatically generated

I then take this and use it in my vectorizer to get rid of any of the words that are not in the “total\_unique\_words” list created above. Code here:

A screenshot of a cell phone

Description automatically generated

First thing I do in the vectorizer is create a numpy array of length train and of width length of the total number of unique words. Next, I loop through the train\_vectorized\_data numpy array of 0s that I created earlier and create a collection of all of the words in that specific review at the respective location in the train data. I add the label on to the list called train\_labels and then I go through all of the words in the review and do not count them (delete them) if they appear less than 10 times or more than 4700 times in the total count of unique words in the whole data set. For example, if the word “the” occurred 5000 times in this data set it would be removed from the list of words in the review. Finally, I add the count of each unique word in the review into the respective location in the train\_vectorized\_data numpy array. I repeat this exact process for the training data to vectorize it as well.

3) I did a bit of hard coding to get the n-fold validation to work. First, I split the whole vectorized training data into 10 equal parts. I then loop through all of those 10 parts, making a different one the test set each time and using the other 9 as the training sets. Here is average score over all the n-fold validation test runs:

A close up of a sign

Description automatically generated

4) For my grid search I used logistic regression and looped through these C values: c\_values = [0.001, 0.01, 0.1, 1, 20, 30, 400] and these penalties: penalties = ['l1','l2']. I ran this with n-fold cross validation again. In the end, this run of the code gave me this output as the best and with this accuracy:

A close up of a sign

Description automatically generated

These values were stored into the best\_c and best\_penalty values which were used in step 5.

5) I ran logistic regression with the hyper params that my grid search determined to be best. The first screenshot below is running logistic regression w/ the solver being liblinear on the test set without touching the hyper params:

A close up of a logo

Description automatically generated

This is running logistic regression of the test data with the hyper params from my grid search:

A close up of a sign

Description automatically generated

As you can see, the test done with the grid search params performed better on the exact same test data by about 0.2% than the default params. This is not a massive improvement (especially because I only ended up changing one param from the defaults because the default C value for logistic regression is 1) but it does show an improvement!