GitHub: https://github.com/caiton1/CS470finalproject

Dataset: https://www.kaggle.com/datasets/colormap/spambase/data

Implementation

This project involves taking in a CSV of word frequencies, capital letter data and finally whether or not it is spam. Each row in the CSV seems to represent a separate email. Due to the context of this project, I wanted to go with naive bayes as it works well with document/text classifications. The model that I am using is called a multinomial naive bayes, as we are using the frequency of words in our data set.

I split the implementation into 4 parts, train, predict, test, remove_correlated as functions.

Train:
This function controls everything related to training minus the set up near the bottom for the function. It works by first splitting the data set into two classes, spam and not spam. Then it takes the prior probabilities P(spam) P(non_spam).

Now, we need some way to do the sum for frequencies. For both data sets, we'll need to sum up the frequencies. Then, take the sum of all frequencies and sum that to make a sum total of frequencies for each data set.

Finally, we calculate the conditional probabilities, using laplace smoothing to avoid underflowing. In this case, I am going to take the sum of word frequencies and divide it by total frequencies to represent probability for each word. Both numerator and denominator will of course have additive laplacing.

The return of this function is the prior probabilities and then an list of probabilities for the spam and non spam dataset, this will be used in predictions.

Predict:
Because of the work done in other functions, predict is a smaller function. All this function does is take the prior probabilities and word probabilities then detect whether or not an email is spam or not. It does so by looking at the frequency of each word and seeing if the word is in the word probabilities list. If it is, it will add to the spam and/or non spam probability cumulatively. Then at the end it does a comparison to see if it's more likely that it's spam or not.

Test:
For this, I measure the predict function for each email/row in the test data set that is passed into it. Then I find the false and true positives here. The area under the curve can also be found here. The result of these metrics will be returned.

Remove correlated:
I found out online that naive bayes works under the assumption that all features are independent. However this is not true in the real world. To better simplify the data set and create more accurate results, we should remove the dependent/correlated features. Here some linear algebra and python math is used.

Performance

In this program I use the test function to handle generating performance reports. The performance for this project seems to be based on 4 factors, accuracy, false positives, true positives and area under the curve.

I am using 5 cross validation. Near the bottom of the code, I split the data into 80% training and 20% test after shuffling it. I then split the training data into 5 “folds” to be individually used against the test data set to get a better idea of how my model performs.

After a run with the uploaded code, this is the return:
SET 0 results:
Accuracy: 0.8219326818675353
False Positive rate: 0.09663409337676439
True Positive rate 0.3257328990228013
Area Under Curve 0.8184981684981685

SET 1 results:
Accuracy: 0.8186753528773073
False Positive rate: 0.09120521172638436
True Positive rate 0.31704668838219324
Area Under Curve 0.8124102564102564

SET 2 results:
Accuracy: 0.8306188925081434
False Positive rate: 0.08794788273615635
True Positive rate 0.3257328990228013
Area Under Curve 0.8258241758241758

SET 3 results:
Accuracy: 0.8197611292073833
False Positive rate: 0.09120521172638436

True Positive rate 0.31813246471226925
Area Under Curve 0.8137435897435897

SET 4 results:
Accuracy: 0.8577633007600435
False Positive rate: 0.0749185667752443
True Positive rate 0.33984799131378934
Area Under Curve 0.8541465201465201

Most of my results are within the 85-90% range of accuracy. This is good and fairly accurate for this type of problem. Looking at other solutions found on spambase (ones that use ML libraries) the accuracy is similar.

The false positive rate is also good, a low false positive is good because we do not want non spam emails being identified as spam (happens in the real world). This means our model does not mistake legitimate emails for spam often

The (relatively) high true positive rate is good, this means that it is accurate at picking up spam emails. The reason that it seems low is that we are *only* considering cases where the email is both spam *AND* has been identified as spam by the model. This isnt considering where the email is not spam and has been labeled not spam by the model.

Area under the curve is also good. A high value means that my model has a good ability to distinguish between spam and non_spam classes and separate them, thus another good metric to use for performance.

Overall, this model seems to be highly effective at identifying spam given this dataset.