Financial Analysis Using Gradient Boosting

Cameron Wat 12/15/2023

Abstract

In this part of the project we will be exploring the Gradient Boosting machine learning algorithm to perform classification and regression.

Gradient Boosting is a machine learning algorithm used in regression and classification tasks. It is known for its high predictive accuracy and is regarded as one of the most powerful techniques for building predictive models. This is because it iteratively combines weak learners into stronger ones, where new models are trained. This lowers the mean squared error of the older model. There are some drawbacks however. Gradient boosting is prone to overfitting and is computationally intense.

In our project, we will be using 2 different datasets to explore Gradient Boosting. The first dataset, and our classification dataset, comes from the University of California, Irvine. The specific dataset focuses on bank marketing and includes 16 different features. The goal with this dataset is to predict if the client will subscribe (yes/no) to a term deposit (variable y).

The second dataset will be our regression dataset, and it comes from Yahoo Finance. We will be doing a case study on Apple's (AAPL) stock price history from the time range 1980-2023 and 01/2023-12/2023. The dataset includes 8 features. Our goal is to use regression to see how well our algorithm predicts the stock price.

Classification

Our classification dataset came with several csv files. We chose to use the one called "bank.csv", containing roughly 4500 data points and 16 features:

"Age";"job";"marital";"education";"default";"balance";"housing";"loan";"contact";"day";"month";"duration";"campaign";"pdays";"previous";"poutcome";"y".

Some of the features we deemed irrelevant and removed them from the training (age, contact, dates). But there are Notable features to know:

default: has credit in default?;
balance: average yearly balance;
housing: has a housing loan?;
loan: has a personal loan?;
poutcome: outcome of the previous marketing campaign
y: has the client subscribed a term deposit?

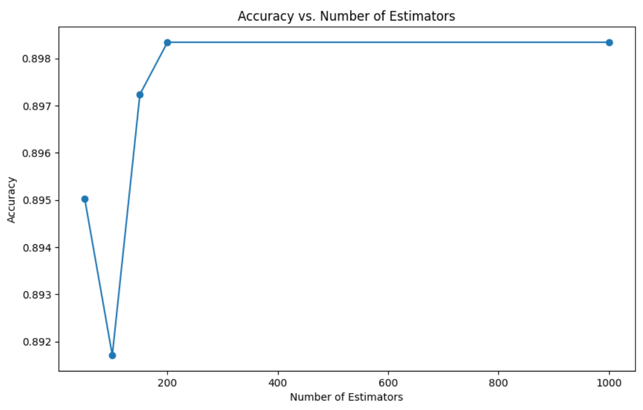
The classification goal is to accurately predict whether or not the clients will subscribe to a term deposit, feature "y".

We first loaded our dataset using the pandas library. Then we encoded the following features. These are what we want to train our model on:

['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'poutcome']

From this, we drop the "y" feature because that is the one we want to classify. For this model, we are using a 20/80 split. 20% of our data goes to the testing set, 80% goes to the training set.

We chose a random state of 42, which initializes the random number generator, that decides the splitting of data into train and test sets. We chose a learning rate of 0.1. This shrinks the contribution of each tree by 0.1 (increased training time, but hopefully higher results). Our max depth was set to 3. The maximum depth limits the number of nodes in the tree. And finally, the number of estimators we used. We selected from [50, 100, 150, 200, 1000]. These are the number of boosting stages to perform. Gradient boosting is prone to overfitting, so a large number usually results in higher performance. After we set our parameters and load in the dataset, we let the SciKit python library handle the rest.



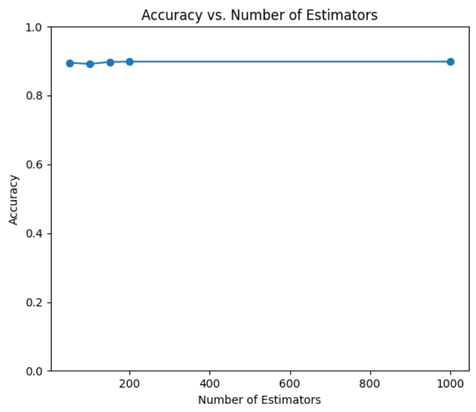


Figure 1: Accuracy Vs. Number of Estimators | zoomed in (left), normal (right)

Our results came back pretty decent. We had a best accuracy of 0.898 or 89.8% over the 1000 estimators. From *Figure 1* (normal), we can see that the overall accuracy was pretty much the same throughout the range of estimators. From *Figure 1* (zoomed in), there was only a 0.003 difference in the accuracy when going from start (50 estimators) to the end (1000 estimators).

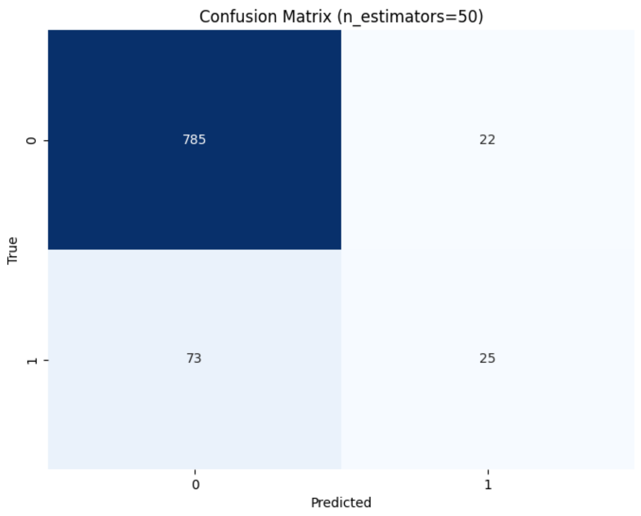


Figure 2: Confusion Matrix (50 estimators)

Figure 2 shows our confusion matrix for the Gradient Boost classification model at 50 estimators. The results are mixed. The model did extremely well classifying clients in the negative class, or those who are not subscribed to a term loan. However, it didn't do as well classifying clients in the positive class, those who are subscribed to a term loan. Below is full report for the confusion matrix in *Figure 2*:

True Negative (TN): 785
The number of instances that were correctly predicted as the negative class (e.g., non-subscription to a term deposit).
False Positive (FP): 22
The number of instances that were incorrectly predicted as the positive class (e.g., predicted subscription to a term deposit, but the true label is non-subscription).
False Negative (FN): 73
The number of instances that were incorrectly predicted as the negative class (e.g., predicted non-subscription, but the true label is subscription).
True Positive (TP): 25
The number of instances that were correctly predicted as the positive class (e.g., subscription to a term deposit).

```
Classification Report (n_estimators=50):
              precision    recall  f1-score   support

          no       0.91      0.97      0.94       807
         yes       0.53      0.26      0.34        98

    accuracy                           0.90       905
   macro avg       0.72      0.61      0.64       905
weighted avg       0.87      0.90      0.88       905
```

The report shows a 91% precision for the no's and 53% precision for the yes's. Further confirming our analysis that our model did better for the negative class. This is not what a good model should look like (both precision scores should be high and around the same). This means that either our model is off, or our data. We will explore the reasoning on why our model was more accurate for the negative class in our conclusion.

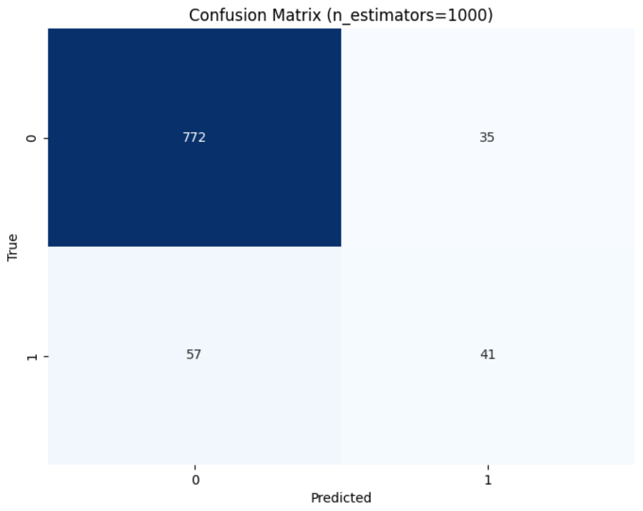


Figure 3: Confusion Matrix (1000 estimators)

Figure 3 shows our confusion matrix for the Gradient Boost classification model at 1000 estimators. The results are pretty much the same as the confusion matrix in *Figure 2*. The model did extremely well classifying clients in the negative class, or those who are not subscribed to a term loan. However, it didn't do as well classifying clients in the positive class, those who are subscribed to a term loan. Below is full report for the confusion matrix in *Figure 3*:

True Negative (TN): 772
The number of instances that were correctly predicted as the negative class (e.g., non-subscription to a term deposit).
False Positive (FP): 35
The number of instances that were incorrectly predicted as the positive class (e.g., predicted subscription to a term deposit, but the true label is non-subscription).
False Negative (FN): 57
The number of instances that were incorrectly predicted as the negative class (e.g., predicted non-subscription, but the true label is subscription).
True Positive (TP): 41
The number of instances that were correctly predicted as the positive class (e.g., subscription to a term deposit).

Our model accuracy for the classification was good, reaching almost 90%, sitting at just 89.8%. However, our negative class precision did 40% better than the positive class (91% vs 53%). How could this be? Upon further analysis and consultation, we realized that we made a mistake during data processing. We didn't make sure we had an even amount of negative and positive class data points. This led to an imbalance during training and testing, and eventually produced the results above.

Regression

For our regression data set, we manually collected our data using the python library yfinance. The data set contains roughly 10,000 data points and 8 features:

Date,Open,High,Low,Close,Volume,Dividends,Stock Splits

Most of these features are relevant to our model, but we did remove a few: date, dividends, and stock split. We removed these because these features don't affect the stock price on a day to day basis. The goal for our regression model is to see how well our algorithm predicts the stock price.

The first thing we did was load our dataset using pandas. Then we separated the features we wanted to train our model on. We chose these features as our training points:

['Open', 'High', 'Low', 'Volume']

We chose these features specifically because the 'Open' is when the stock price for the day is first logged (and is usually, not always, the close price from the previous day), the 'High' and 'Low' is where the price of the stock hits its peak and trough for the day. And for 'Volume', we felt it was important to use because it tells us how much of the stock is being traded. We were curious to see if large or low volume days contributed more or less to the price. Our target feature was ‘Close’. We wanted this to be the target because that is when the price is final for the day. This model will have the same parameters as the classification model.We are using a 20/80 split. 20% of our data goes to the testing set, 80% goes to the training set. We chose a random state of 42, which initializes the random number generator, that decides the splitting of data into train and test sets. We chose a learning rate of 0.1. This shrinks the contribution of each tree by 0.1 (increased training time, but hopefully higher results). The number of estimators we used are also the same, [50, 100, 150, 200, 1000]. After we set our parameters and load in the dataset, we let the SciKit python library handle the rest.

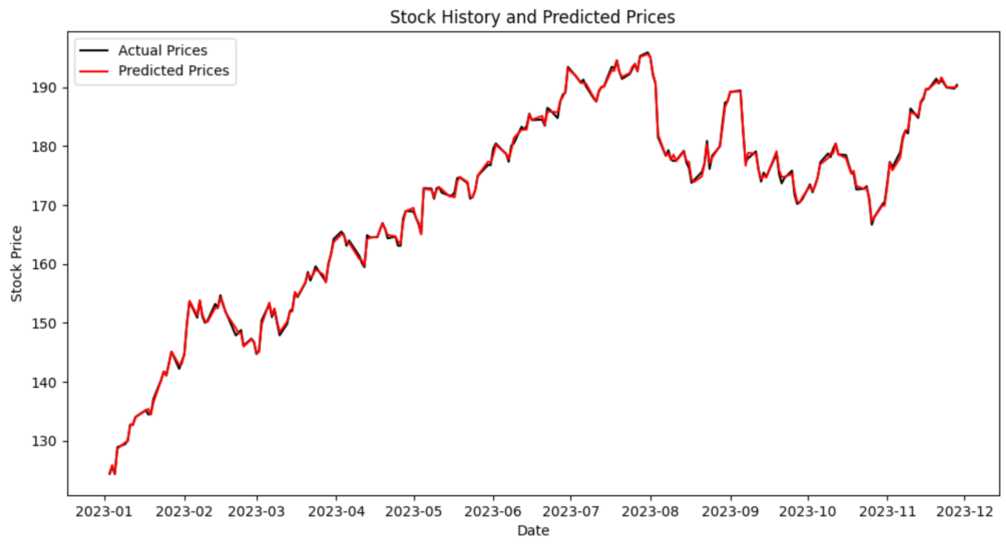


Figure 4: Stock Prediction Using Gradient Boosting (01/2023 - 12/2023)

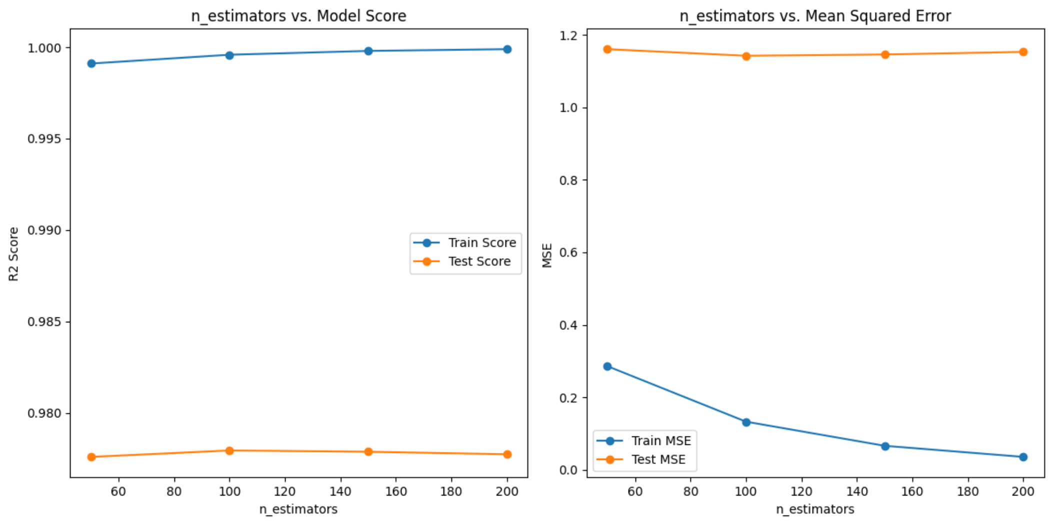


Figure 5: Accuracy and MSE vs Estimators (01/2023 - 12/2023)

Our results for the year to date range (01/2023 - 12/2023) was exactly what we are looking for. As we can see from *Figure 4,* our gradient boosting model was near perfect when predicting (in red) the actual stock price (in black). Our best accuracy was 99.9%, near perfect. On the left plot in *Figure 5*, the train score increases over time while the test score peaks at 100 estimators. On the right plot in *Figure 5,* the train MSE decreases and the test MSE stays relatively flat. Indicating that the model adjusts its parameters to minimize the difference between its predictions and the actual target values of training data. Below are the actual numbers from *Figure 5*:

```
Train score: 0.9998894165554167
Test score: 0.977956603325492
Train mse: 1.1532338165769063
Test mse: 0.28576241537061137
```

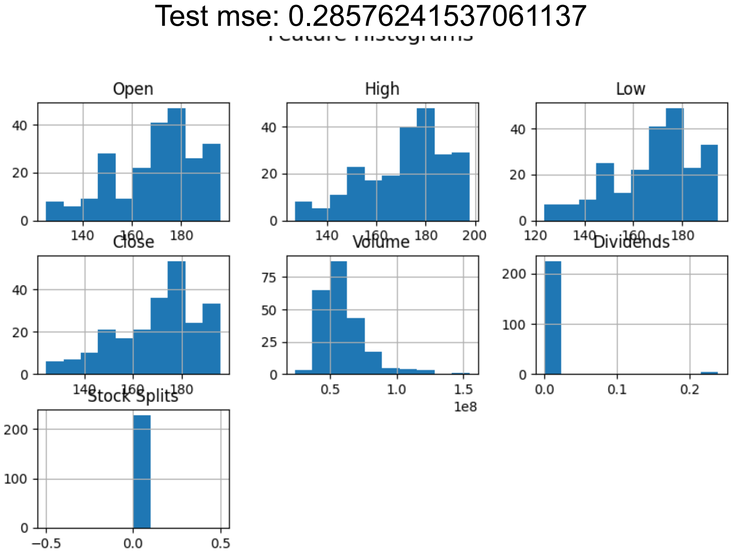


Figure 6: Number of Samples in Each Class (01/2023 - 12/2023)

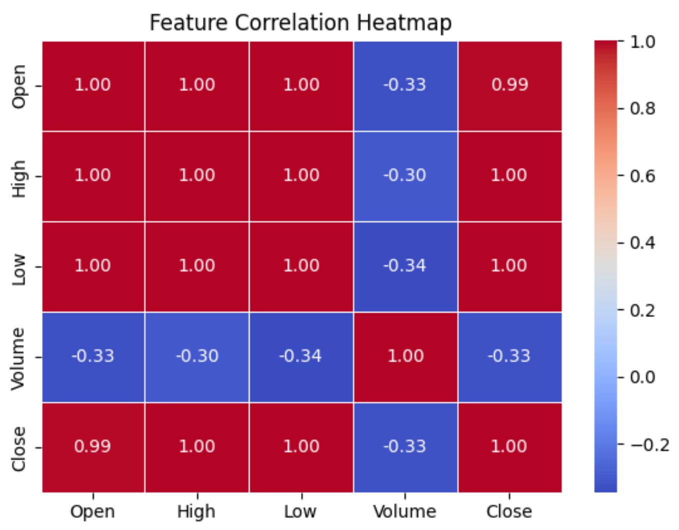


Figure 7: Feature Correlation Heatmap (01/2023 - 12/2023)

Figure 6 and Figure 7 are extra information about the data set and how features are associated between the different variables in a dataset. In *Figure 6,* we can see the prices (top row) are all mostly from the 160+ range. In *Figure 7,* according to our model, all the features except the volume were positively correlated. The volume feature was the only one with a negative correlation which is interesting.

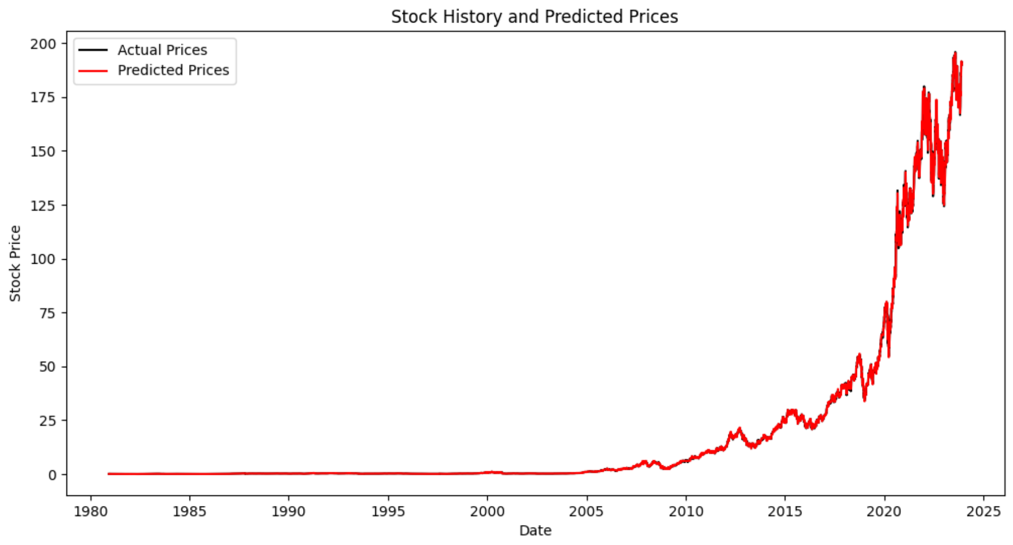


Figure 8: Stock Prediction Using Gradient Boosting (01/1980 - 12/2023)

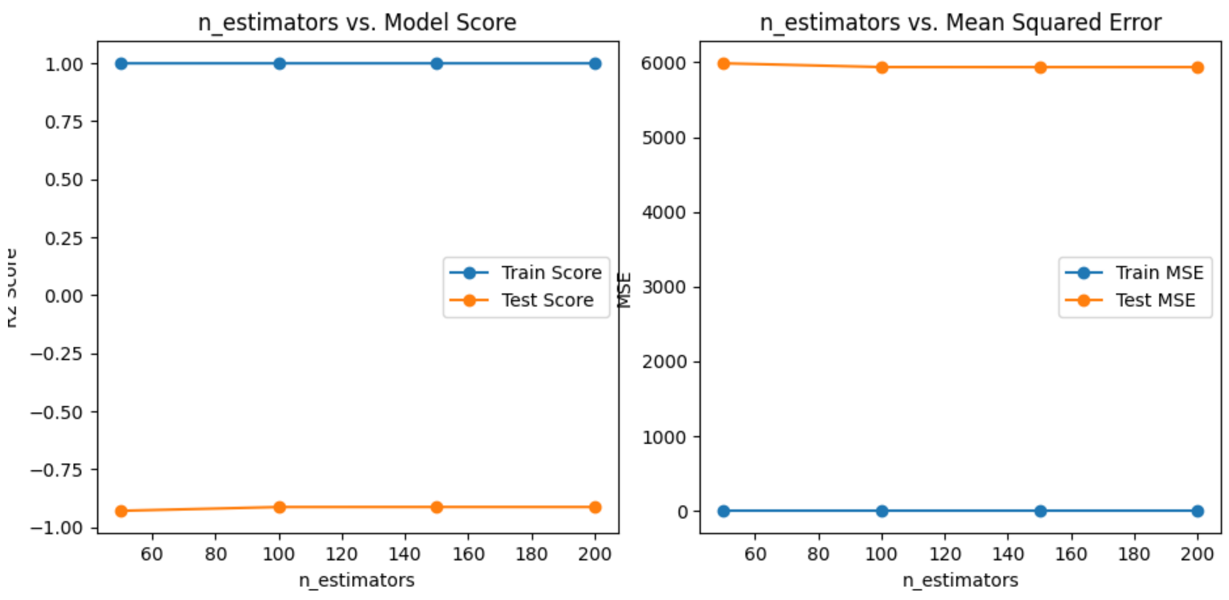


Figure 9: Accuracy and MSE vs Estimators (01/1980 - 12/2023)

As we can see from *Figure 8*, we had similar results to *Figure 4*. Our gradient boosting model was near perfect when predicting (in red) the actual stock price (in black). Our model's best accuracy was also 99.9%.

In *Figure 9,* the train accuracy (left plot) looks to be a constant 0.99. However, the test accuracy was in the negatives. Our MSE for the all time data (01/1980 - 12/2023) was off. We achieved a near 6000 train MSE score, and a 0.0037 test MSE score. The train MSE score is not what we should get. Below is the scores for *Figure 9:*

```
train score: 0.9999508877292158
Test score: -0.9123326614874661
Train mse: 5987.821099685426
test mse: 0.0037118326757651534
```

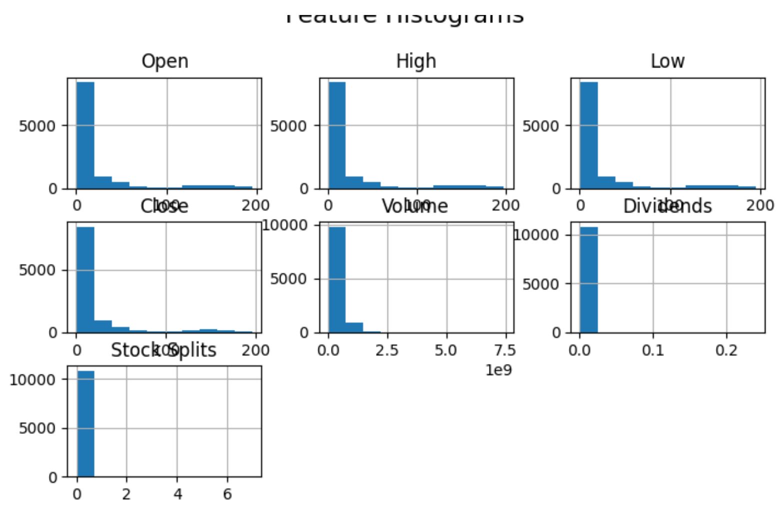


Figure 10: Number of Samples in Each Class (01/1980 - 12/2023)

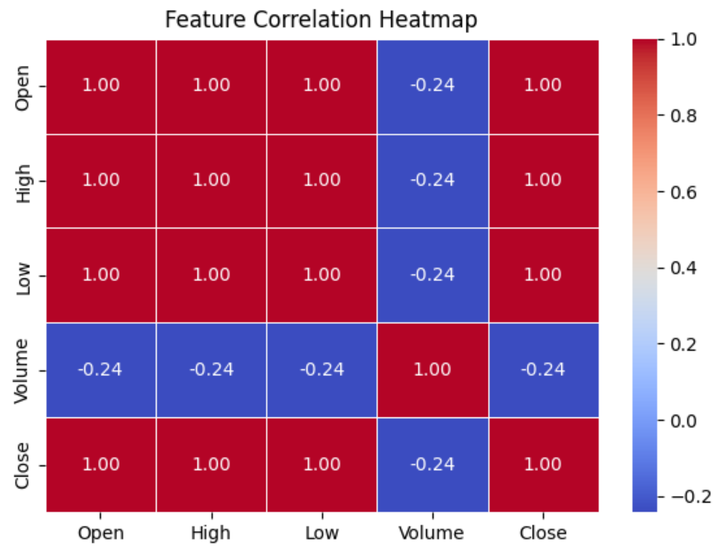


Figure 11: Feature Correlation Heatmap (01/1980 - 12/2023)

Figure 10 and *Figure 11* are extra information about the data set and how features are associated between the different variables in a dataset. In *Figure 10,* we can see the prices (top row) are all mostly near 0-20 range. In *Figure 11,* we had the same correlation as the year to date dataset in *Figure 7.*

Our model had very high accuracy when predicting stock prices. Each of the time frames (01/1980 - 12/2023 and 01/2023 - 12/2023) produced a 99.9% accuracy. Both time frames had near identical heat map correlations, where the volume had a negative correlation and everything else a positive correlation. Our MSE scores for the year to date time frame (01/2023 - 12/202) yielded better and more realistic results than the 01/1980 - 12/2023 time range. A 6000 MSE score could mean that our model was overfitting, and capturing noise/outliers. In *Figure 10,* we see a massive bar between the 0 and 20 range, indicating that there were way more data points there than anywhere else. When we compare *Figure 10* to *Figure 6*, *Figure 6* was more spread out. This could be the reason for the 6000 MSE score we achieved in the 01/1980 - 12/2023 time range. Overall, still a successful model.

Conclusion

Our exploration with the Gradient Boosting machine learning algorithm for both classification and regression tasks yielded high results. There are areas we can improve on however. Firstly, we can improve on how we process and handle our data. From both the classification and regression, we did not properly clean the data for outliers and we did not make sure we had an even amount of classifiers. This led to results being way off and skewed in one way. Another area we can improve on is exploring other features or removing features we used. Playing around with the different features will allow us to see if our results yielded differently. Despite some improper data handling, our models did well accurately classifying and predicting stock prices.