Neel Raut and Andrew Sun 12/8/2023

CS4100 Project Final Report

Relevant project files can be accessed at the following GitHub link.

Describe the problem on a high level, including some motivation for choosing it (just like in your presentation).

A high-level interpretation of our problem is that in the versatile and spontaneous environment of modern-day financial markets, one seeks a way to “game the system” and somehow be able to predict the moves in the market before they even happen. Now, this is not something that is guaranteed for every person and certainly isn’t danger-free - but therein lies the inherent risk of prediction. Ideally, this is minimized when possible as knowledge of the market compounds over time. And so the dilemma arises: how can past and present data of the market be utilized and combined in such a way to optimize predictions in the volatile market such as to maximize future profit and minimize cumulative risk? This project attempts to solve just that, by trying to create an algorithm that can take advantage of market trends to churn out a profit making investment strategy. This fits in the larger field of algorithmic trading: a complex & nuanced application of AI/ML with statistics in pursuit of optimized algorithmic solutions. We both found this field to be very interesting and saw this project as a great opportunity to advance our skills in algorithmic trading while applying the knowledge we’ve learned so far.

Concrete technical problem statement, model (if applicable), inputs/outputs.

Our problem statement, or rather engineering goal, is to develop a machine learning model with the capability to provide trading insights based on past and current market data, utilizing economic and traditionally market-related indicators.

This requires a two-pronged approach. First, we aim to make use of a logistic regression model to predict the short-term longevity of selected stocks; we want to take a stock and determine if there is potential for growth in terms of value and how much that scales up. Then, we want to formulate a decision-making algorithm that will use these results and their probabilities and figure out the optimal policy: the ideal stocks to buy, hold, and sell at this point.

Considering the unpredictable nature of many individual stocks on the market, we opted to consider a stable stock index: the S&P 500. Though its growth isn't perfect, it reacts relatively in sync with changes in the economy/world and is a good reflection of the market throughout history. We hope that by using something as stable as the S&P 500, we will be able to train our model to be more accurate and reliable. Our output from the model will be the probabilities of the growth for each stock, and our output from the algorithm will be an optimal set of actions to take such as to strike a rich balance between profit and vulnerability.

Something to note is that we are carefully tailoring our project so as not to simply apply an existing algorithm to an existing dataset, but rather toy and experiment with effective ways to go about hypertuning our parameters for our model. A large chunk of making our model better lies in thought-out research/analysis on market data to grasp trends efficiently and effectively. Specifically, when drawing out trends from the S&P data, we want to use some combination of existing features to create new ones that will help synthesize new indicators or increase the relevance of others - similar to what we did in ex4/pa4. Say, for example, that we are in a market where interest rates are at an all-time low, and unemployment is near 3% (lower end of the ideal range for a healthy economy)! In such a case, we would use the batch version of stochastic gradient descent to compare indicator performance and come to a conclusion about growth of a certain stock, which would likely be higher. Whereas on the flip side, with high interest rates and unemployment, the prediction for growth of that same stock would likely be lower. This highlights the importance of the use of parameters in affecting our final outcome.

Another noteworthy component when it comes to deterministic features is disaggregating indicators of contrasting frequency. Namely, many features are only available in monthly granularity, whereas the historical price data is provided on the daily (yes, we know it can be aggregated over a month, but it's the sudden daily changes that really lead to the fluctuations in stock potential). Since many of the economic indicators are month-based, figuring out how to effectively use this data in a daily-granularity learning model is another challenge we will tackle.

Describe the dataset you used, if applicable.

The current datasets we are using are daily historical prices of the S&P 500 Index, excluding weekends and federal holidays, or any such occasion that would cause the US markets to be closed on a particular day. This dataset included information about the open, close, high, low, and trading volume of the S&P 500 from January 3, 2000, to December 6, 2023.

Methods / algorithms that you used or developed. If you are using algorithms not covered in the class, provide a description of it so we know you understand what you have been using.

We used a logistic regression model to classify stock growth potentials. We engineered features (based on the pre-existing data) like the Simple Moving Average (SMA), Upper Bollinger Band (UBB), Lower Bollinger Band (LBB), 52-Week High, and 52-Week Low as our input variable, and the difference in closing price of the S&P 500 on a day-to-day basis as our target variable.

We then expanded upon our model to design an algorithm which would determine the best action to take. We used the model's predictions to generate log-likelihood probabilities. By doing so, we were able to obtain a probability distribution from which we set thresholds for when it would be ideal to buy/sell, and went on to classify each date's ideal action.

We also made another, rather simpler algorithm, for this same purpose. This algorithm looked at the change in predicted values and created buy/sell signals based off of that, combining the two values to create a general signal. This signal dictated the action chosen by this algorithm, based on, again, a buy and sell threshold.

If you are basing your project on someone else's work, explain on a coarse level what they have done, and if you are doing anything different.

Referenced the following link to come up with the functions for the features used in iteration 3 of the logistic regression. This person utilized many different features which we wanted to play around with in our third iteration so we used them, though not in the same way as he did, as we simply incorporated them into our pre-existing list of features while they did a different smoothing/scaling process than we did. With more time, we likely would have come up with a similar approach as they did, perhaps increasing our test accuracy.

Empirical results, including details on the experimental setup (be precise about what settings /data you ran experiments with). What were your hypotheses / what do you expect to see from your experiments?

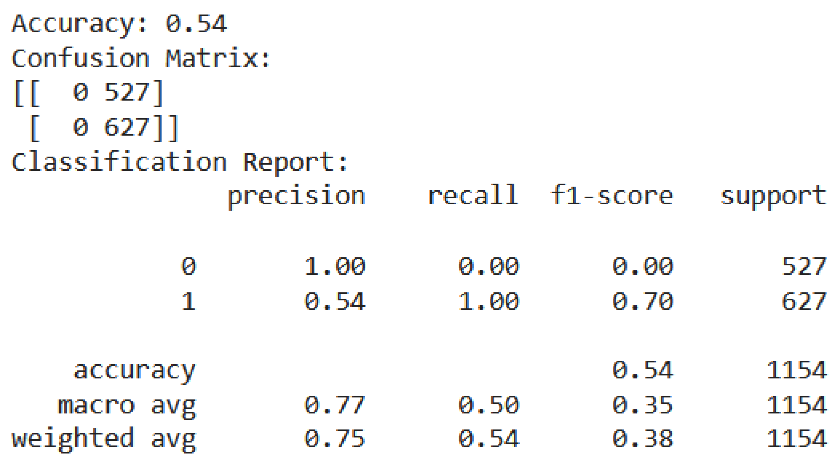
A large portion of our time went into learning about different features that serve as ideal indicators in a market, and then experimenting with said features for the logistic regression model to find what worked and what didn't work. One thing that didn't work was the economic indicators; we felt that, especially within a daily time-frame, longer scale statistics such as inflation, unemployment, or interest weren't as relevant. Any effect these do have are likely incorporated in some way into other variables, such as trading volume.

We built a Pandas dataframe using raw data about the S&P 500 as mentioned before, and cleansed this data by dropping missing/incomplete values. Then, we determined our features variable X and target variable y. Our features changed through different iterations of the logistic regression model, starting with just using the plain inputs from the dataset (excluding date), to then using engineered features based off of the plain inputs, to then appending much more complicated and multi-layered features. However our target variable remained the same throughout: it was a binary classifier that classified a day into either group 1 (the close price of the next day is greater than the close price of the current day) or group 0 (the close price of the next day is less than the close price of the current day).

We had a 80/20 test/train split, meaning that we trained on 80% of the data and tested on 20% of the data. After each iteration of logistic regression, we analyzed the accuracy, confusion matrix, and a classification report of the precision, recall, f1-score, and support. This helped us determine whether we were moving in the right direction or not. What we hypothesized to see after each iteration was an increase in accuracy and better and better confusion matrices. We expected to see tighter weights that would serve as a general benchmark for predicting the target variable.

Finally, we worked on the algorithms that would determine the ideal action to take on a day-to-day basis; one algorithm used the log-likelihood function whereas the other was a difference in predicted values (linear). For the log-likelihood algorithm, we hypothesized that what we'd see in the end as the ideal actions would vary as the log-likelihood was based on a set threshold that could rapidly affect the determined actions. Likewise, the prediction-based algorithm was based on predicted values, and an arbitrary threshold for a signal, so we didn't know what to expect there either.

Iteration 1:
As a baseline, we used only data within a current day to predict the state of the next day. This meant simply running logistic regression on {Open, Close, High, Low, and Volume}. We did not expect this to have a great degree of accuracy - it's imperative to draw trends from previous days to predict future outcomes, which this does not do. Still we thought it might serve as a useful baseline test to compare future feature sets to.

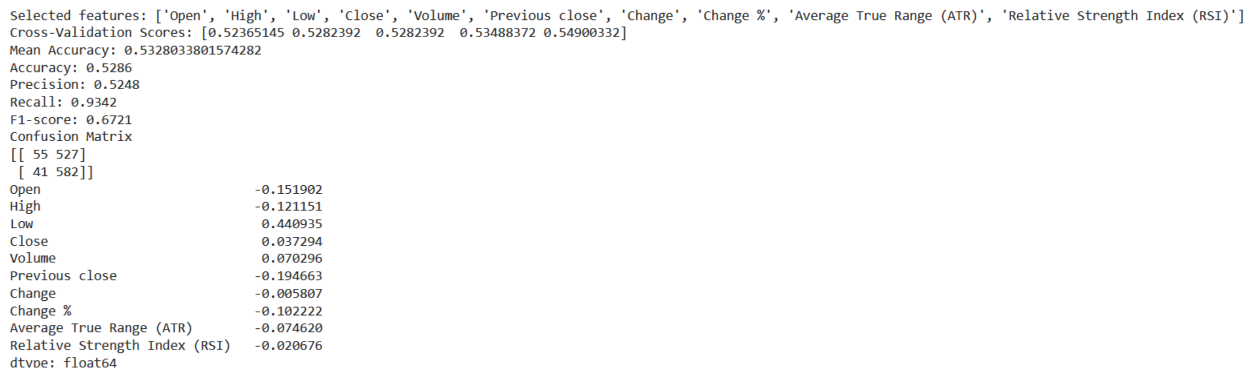


```
Accuracy: 0.54
Confusion Matrix:
[[  0 527]
 [  0 627]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.00      0.00       527
           1       0.54      1.00      0.70       627

    accuracy                           0.54      1154
   macro avg       0.77      0.50      0.35      1154
weighted avg       0.75      0.54      0.38      1154
```

The output of this model suggests that it had trouble creating conclusive weights with an accuracy reasonably better than guessing - this is as expected - as we stated earlier, learning to draw trends and incorporate those into ML models is going to be more effective - and the focus.

Iteration 2:

Our second trial involved incorporating trends into the regression model. For this one, as features, we added {Change (from the previous day), Change Percent, ATR (average true range), and RSI (relative strength index)}. We expected a change in test accuracy, since the model now incorporates more data, and uses information about recent pricing trends.



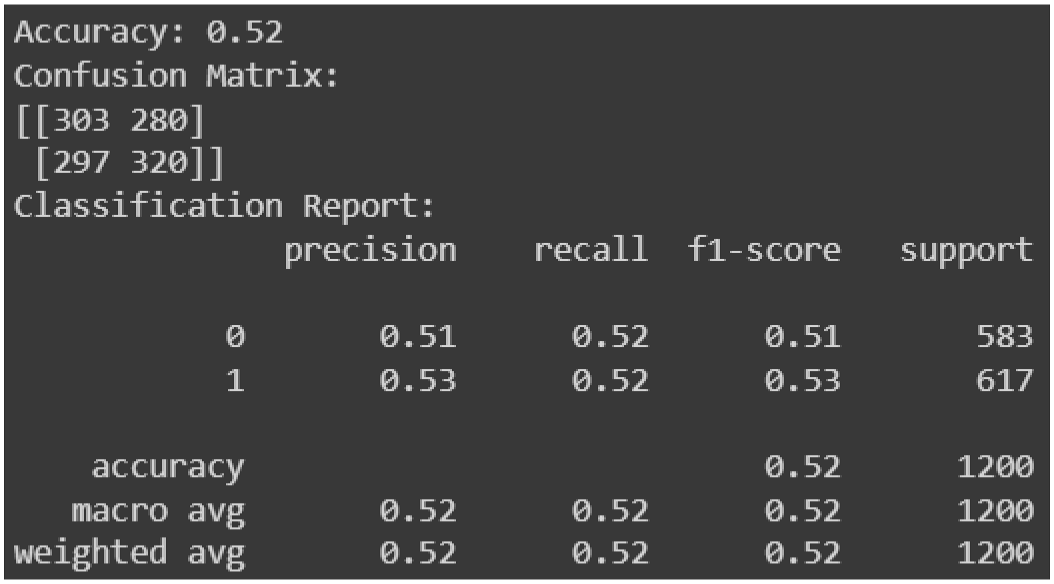
```
Selected features: ['Open', 'High', 'Low', 'Close', 'Volume', 'Previous close', 'Change', 'Change %', 'Average True Range (ATR)', 'Relative Strength Index (RSI)']
Cross-Validation Scores: [0.52365145 0.5282392  0.5282392  0.53488372 0.54900332]
Mean Accuracy: 0.5328033801574282
Accuracy: 0.5286
Precision: 0.5248
Recall: 0.9342
F1-score: 0.6721
Confusion Matrix
[[ 55 527]
 [ 41 582]]
Open                            -0.151902
High                            -0.121151
Low                              0.440935
Close                            0.037294
Volume                           0.070296
Previous close                  -0.194663
Change                          -0.005807
Change %                        -0.102222
Average True Range (ATR)        -0.074620
Relative Strength Index (RSI)   -0.020676
dtype: float64
```

As visible from the training and test results, the model has not changed much in accuracy. We identified two problematic areas for refinement for the next set of features. First, more can be done with the technical indicators rather than simply adding them as features. Second, we noticed possible multicollinearity in the variables affecting the model based on our experiments and research. Based on our research, multicollinearity can be a common issue within trading algorithms, where although features/variables may be different, they present the same information - a type of overcounting important to avoid. We added the following code to eliminate variables that in this set of features are non-independent:

```
for col in data.columns:
    if col not in ["Date", "Volume"] and all(abs(corr[col][other]) <= 0.9 for other
in data.columns if other != col):
        selected_features.append(col)
```

Iteration 3:

Our third trial involved adding more features, and experimenting with how to draw additional information from them. We used {'SMA_200', 'Upper_Band', 'Lower_Band', '52_Week_High', '52_Week_Low', 'EMA_12', 'EMA_26', 'RSI', 'Stochastic_Oscillator', 'Williams', 'MACD', 'Price_RoC', 'OBV'} - this is selection of technical indicators to use as features. We initially believed that additional features might help increase our accuracy, but it didn't create a change like we thought.



```
Accuracy: 0.52
Confusion Matrix:
[[303 280]
 [297 320]]
Classification Report:
              precision    recall  f1-score   support

           0       0.51      0.52      0.51       583
           1       0.53      0.52      0.53       617

    accuracy                           0.52      1200
   macro avg       0.52      0.52      0.52      1200
weighted avg       0.52      0.52      0.52      1200
```

As visible from the training and test results, the model has not changed much in accuracy again, as the most effective algorithms will take technical indicators in context to draw trends from them.

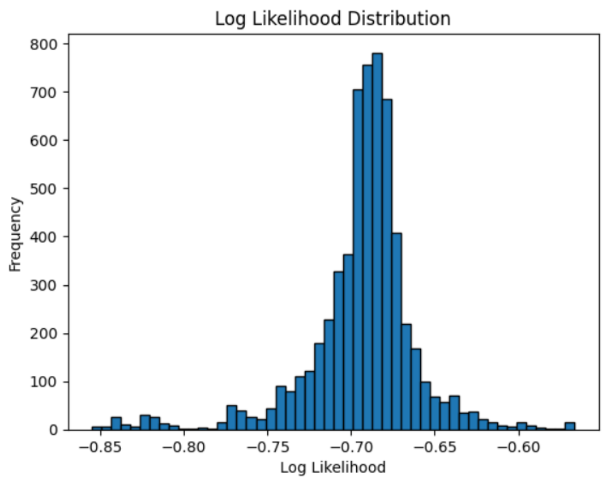
For example, RSI - the relative strength index, measures speed and change of price movements. Typically, when the RSI rises above 30, it is a bullish sign and when it drops below 70, it is a bearish sign. When it fluctuates but stays within these bounds, it can signal upwards or downwards trends. Additionally, RSI may diverge from real price trends - another indicator with nuanced real-world significance. We attempted to capture additional data from technical indicators and their trends, derivatives, etc. For example:

```
for r in range(len(rsi_values)):
    temp = rsi_values[r] - rsi_values[r - 1]
    if rsi_values[r] > 70:
        rsi_dx.append(temp)
    elif rsi_values[r] < 30:
        rsi_dx.append(-temp)
    else:
        rsi_dx.append(0)
```

attempts to use daily change in RSI, along with RSI value to represent one feature. However, we found that such features were difficult to succinctly translate to code and implement.

Algorithm 1 (Log-Likelihood):

After training a logistic regression model, we then wanted to create a trading algorithm that is able to make decisions based on the model classifications and probabilities.



```
        Close  Predicted  Log_Likelihood Action
0     1455.22        NaN             NaN   Hold
1     1399.42        0.0       -0.702043   Hold
2     1402.11        0.0       -0.703493   Hold
3     1403.45        0.0       -0.703892   Hold
4     1441.47        0.0       -0.701362   Hold
...       ...        ...             ...    ...
6016  4567.80        1.0       -0.685186   Hold
6017  4594.63        1.0       -0.683636   Hold
6018  4569.78        1.0       -0.680935   Hold
6019  4567.18        1.0       -0.677977   Hold
6020  4549.34        1.0       -0.674754   Hold
```

So, we took the predicted values and found the log-likelihood probabilities distribution based off of those values. We graphed the distribution to display the frequency of the log-likelihood values and establish a visual from which we could set the buy/sell thresholds. Noticing that this was an approximately Gaussian distribution, we extracted the mean and standard deviation and set our bounds accordingly, with the "buy" threshold being one standard deviation above the mean and the "sell" threshold being one standard deviation below the mean. Thus, anything within one standard deviation of the mean would be considered a "hold".

Algorithm 2 (Change in Prediction):

Although we now have a great log-likelihood probability-based algorithm, we also wanted a simple algorithm that did something similar to how we set up our target variable in logistic regression. In other words, we wanted to see how the predicted prices themselves (not the probabilities) would stack up when deciding actions. So we developed this algorithm to do just that, creating buy/sell signals based on how the previously predicted values differed and the summing that up to create a general “signal”. This was used as the basis for which to classify our “buy”, “sell”, or “hold” actions.

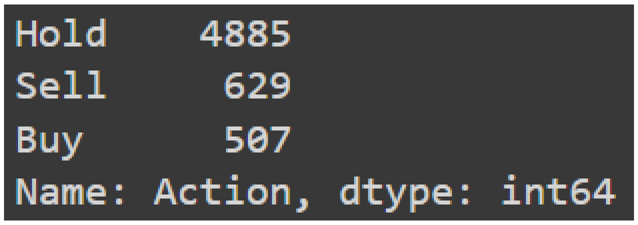
Analysis of empirical results. What worked, what did not, and why?

From our first iteration of logistic regression, we noticed that our confusion matrix was slightly off, in that it didn’t classify the data into class “0” (closing price decreased between two days) all the time. We analyzed the weights of the model and found them to be quite small (near 0) and thus believe this was why the model couldn’t classify correctly. However we did get an accuracy that was slightly better than guessing, implying that there was indeed some merit behind conducting such an analysis.

In our second iteration of logistic regression, we wanted to improve our feature set in the hopes of boosting accuracy and yielding more accurate predictions such as to make a more sensical confusion matrix. We engineered some custom features based on existing market indicators: the Simple Moving Average (over 200 days), Bollinger Bands, and 52-week High/Low. The results showed that the accuracy was relatively unchanged, and was actually 1% lower (likely due to the introduction of said features). A major problem we uncovered here was the presence of multicollinearity: the relation of different independent variables which we believe played a role in our low test accuracy. However we did output a confusion matrix from which we could understand more about how the data was being classified, and verified that now, the data was indeed being classified in the right way (diagonals/off-diagonals correlate to the accuracy).

In our third iteration of logistic regression, we wanted to beef up the test accuracy one last time to see if we could break a higher test accuracy range. This is harder said than done, as the aforementioned volatile environment of the market makes prediction not much better than random guessing at times, so we sought to improve our feature set one more time in order to push the model to predict more accurately. We did indeed use many features, but noticed no major gain in test accuracy. It's unfortunate, but we believe that we are just missing the context of these indicators, and incorporating that somehow into our features may increase the potential testing accuracy of our model. After all, it's not just adding a bunch of features but also relating them that does the "magic" behind logistic regression. However we did notice that using this third iteration did increase the buy/sell bracket of the two learning algorithms below, likely because of increased nuance in predictions and the related log-likelihood distribution.

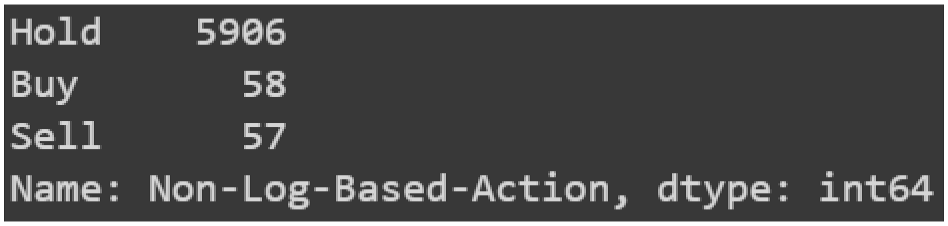
Now, after working our way through the logistic regression model, we devised an algorithm that would take in the log-likelihood probability distribution and output actions that should be taken in accordance with these probabilities. When graphing the log-likelihood distribution we noticed it was approximately normal so this allowed us to make buy/sell thresholds based on the mean and standard deviation. Specifically, we had the buy threshold be one standard deviation above the mean and the sell threshold be one standard deviation below the mean.



```
Hold    4885
Sell     629
Buy      507
Name: Action, dtype: int64
```

As can be seen from the summary of the actions generated by this algorithm (based off of iteration 2 of log-reg), we had a majority "Hold" and roughly equivalent "Sell" or "Buy". Again, this makes sense looking at the distribution of the log-likelihood function, and the majority of actions being "Hold" corresponds to the general idea of the S&P 500 index being a "long-term investment" as it is quite stable and tends to give out positive return over time, recovering from temporary downturns in the market. This could obviously be altered by setting different buy/sell thresholds, and that would all depend on the specific person's risk tolerance. Those who may be more risk-inclined would narrow the interval of the "Hold" action while those more risk-averse would increase this window to lessen the amount of buying/selling in case of some unfortunate event in the future.

Though the log-likelihood algorithm was great and all, we also wanted to devise an algorithm that would determine actions in a similar fashion to how we implemented a binary classifier for our logistic regression model. Specifically, we wanted to look at change in predictions as a way to determine the appropriate actions by date. What we did here was simply look at the values predicted by the model and create buy/sell signals that would look at previous predictions and "light up" based on if there was a viable change. These signals would then be concatenated into one general signal which would serve as the basis for our action classifier. We used signals of -1 (sell), 0 (hold), and 1 (buy), serving as our essential "thresholds."



```
Hold    5906
Buy       58
Sell      57
Name: Non-Log-Based-Action, dtype: int64
```

As can be seen from the above (also based off of iteration 2 of log-reg), our actions were significantly biased towards "Hold," especially compared to our results from the first algorithm. This shows a highly conservative investment strategy, but again matches up with the idea of the S&P 500 being a "long-term investment." These results were very interesting as they highlighted the difference between comparing the base prediction values and actually calculating the log-likelihood values. Both algorithms yielded viable investment strategies, and represent different ways to go about analyzing predicted data to come up with optimal financial strategies.

Discussion / future directions (if any). What difficulties did you encounter, if any? Did anything not go according to plan? If you had more time to spend on the project, what would you have liked to do next? What advice about the project would you give to future CS 4100 students?

Our biggest challenge throughout the project was finding the right features to use to balance accuracy with practicality. We wanted to use the technical indicators that would provide the most information so as to efficiently and effectively classify our target variable: stock potential. As evidenced by our multiple iterations, this was a fairly difficult task due to the immense domain knowledge required to truly understand these features along with the explosive nature of the market. If we had more time to spend on the project, we would have taken a much deeper look into the major technical indicators used to analyze the market (the likes of which many major hedge funds/trading companies like Citadel, Goldman-Sachs, 2-Sigma use) and the relation between them (keeping multicollinearity in mind). With enough research, we could hopefully tailor our features as such to make better predictions.

Another challenge lied in the process of designing our action classifier algorithms. A large impediment here was setting ideal bounds as there were no specific mentions as to how to find the ideal thresholds. After analyzing and experimenting with different thresholds we came to the conclusion that said thresholds were based simply on risk tolerance, as market value of the S&P 500 varied by the day and so predicting ideal actions isn't so straightforward. If we had more time to spend on the project we would have looked into some of the automated threshold mechanisms to find the optimal sequence of actions, diving into scikit-image or OpenCV to create optimal thresholds from histogram data.

Stock trading algorithms and capturing patterns out of the seemingly random nature of the market is a difficult task. I think that with more time for this project, we could have crafted a varied and effective feature set that would definitely yield larger payoff in terms of model classification accuracy and thus predictions - leading to more optimal actions. For those future market-driven CS4100 students willing to take on similar feats, our advice would be to really take the time to understand the inner workings of the market and understand that though we can continually develop more and more accurate models/algorithms, the nature of the market is inherently volatile, and what's important is coming up with the features (technical indicators) that maximize the accuracy rather than the final accuracy itself.