Jaswanth Sai Pyneni | jpyneni3

# Q-Learner Strategy Report

## Overview:

The purpose of this project was to develop a Q-Learning Trading Agent.

## Trading Problem to Learning Problem:

I had to take multiple steps in order to frame the trading problem as a learning problem for the learner. This strategy learner, implemented in the StrategyLearner class has two methods, addEvidence() and testPolicy(). The purpose of addEvidence() is to allow to the trading agent to iterate through different states and possible actions, as a Q-Learner does, in order to learn a policy. The purpose of testPolicy() is to examine how effective that the policy learner by the Q-Learner is.

During the learning phase, the addEvidence() method created a prices dataframe, representing the day to day close price for a certain specified symbol in specified time period. Then, using the indicators code written for the manual strategy project, the method creates a dataframe for the Bollinger Band values for that stock on every date within the time period and then another dataframe representing the MACD values for that stock on every day within the time period. These indicator values, for any given day, represent that "state" of the stock that day for my learner and based off those states, the learner outputs one of three actions – buy, hold, sell. This is the core of how I framed this trading problem as a learning problem- indicator values on a certain day as states and the potential market trades that should happen because of those values as actions.

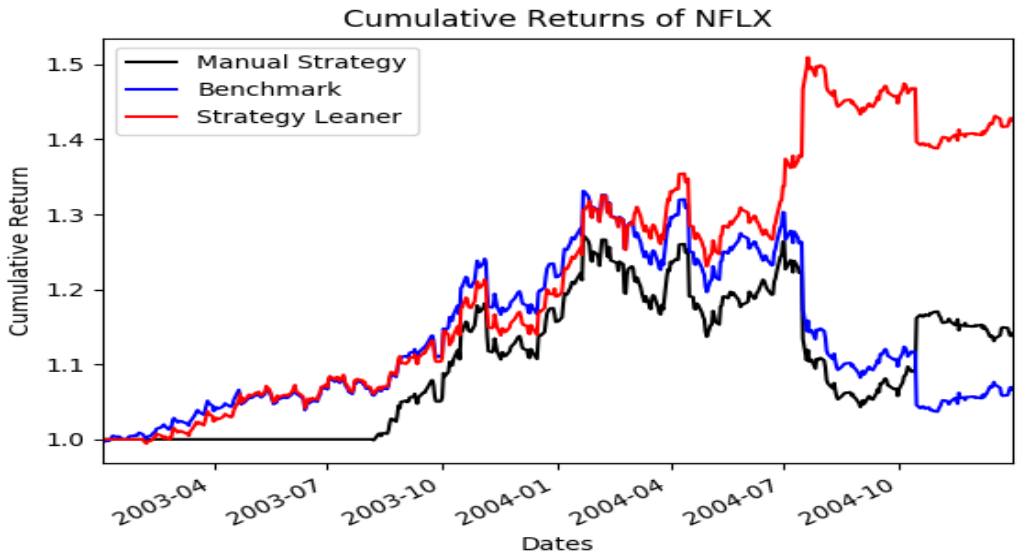
In order to create states from the indicator values such that the states are representative of all three indicator values (upper Bollinger Band, lower Bollinger, and MACD) as one comprehensive state, I had to normalize and discretize the indicator values. First, to combine the upper and lower Bollinger Band into one cohesive unit, I calculated the Bollinger Band Percent Bandwidth (%B) using this formula: %B = (Price - Lower Band)/(Upper Band - Lower Band). The Bollinger Band percentage presents the same information as having two distinct bands because with Bollinger Band percentages, these relations exist: if price is at the upper band then %B equals 1, when price is at the lower band then %B equals 0, when price is above the upper band %B is above 1, and when price is below the lower band %B is below 0. This derivation from regular Bollinger Bands helped to create one representative value for that indicator. Then, to convert the %B values into integers on a limited-scale (to represent possible states), I made 10 bins, with each bin getting all the data points in that decile. I created the decile groups by starting the first bin at the minimum value of the %B values, ending the last bin at the maximum value, then using a step size of (max-min)/10. This process allows me to present any value of %B using integers 0-9. Then, I focused on making discrete states out of the MACD indicator. As my MACD signal in manual strategy was based on the zero-line crossover, with a positive MACD value today and a negative value yesterday indicating a buy, and actual value of MACD being negligible with how I used it, I converted all positive MACD values to 1 and all negative values to -1. Then, I subtracted yesterday's value from today, resulting in either 2, 0, or -2. In this case, if today's value is two, that means that today's MACD was positive and yesterday's was negative. Then, to get rid of negative numbers, I added 2 to every day so after all this, every day's MACD state was either a 0, 2, or a 4. At this juncture, I had all my Bollinger Bands represented by integers 0-9 and MACD represented by 0,2 and 4. To combine those two into one unique state, that takes both values into account, I added the %B state and MACD state together as strings to create 100 possible states, of which at most 30 would be used because the number in the ones place would only every be a 0, 2, or 4.

After this point, I had a data frame with a combined indicator value for each day, representing the state of the market for that stock on that day. Using this for states, I then, in addEvidence() trained my Q-Learner agent by iterating through each day and querying the QLearner with the state of that day and the reward for being in that state on that day. I calculated reward by multiplying today's stock holdings times daily return of today (value of stock from yesterday to today) times one minus self impact. For each day, depending on what action the Q-Learner query outputted, I updated a trades data-frame with buy or sell orders and updated current stock holdings as necessary. I did all of this in a while loop that runs for either 500 epochs or at-least 10 epochs and converged values (the trades data-frame does not change in that epoch). Then in testPolicy() , I use the policy to create the trades data-frame that the Q-Learner trading agent gives by iterating through each day in the indicators data-frame and calling querysetstate() on the state of every day to decide what kind of trade should be made on that day, if any. In testPolicy(), I made sure to set the random action rate of the Q-Learner rate to 0 to ensure that every outputted state is from the actual Q-Learner's learnings and not random.

Experiment 1:

To test the effectiveness of my Q-Learner trading agent, I trained and tested my Q-Learner on NFLX from January 1st 2003 to December 31st 2004. This was the same in-sample period that I used to test my manual strategy in that previous project. In order to make a trades data-frame out of the output from testPolicy(), which only outputs a data-frame that has either -2000, -1000, 1000, or 2000 for each date, I made an Orders column that says Hold be default, and SELL whenever the testPolicy dataframe has a negative number (because that's what that means) or a BUY whenever the testpolicy dataframe has a positive shares number. Then, I took the absolute value of each number of shares value to basically manipulate the testPolicy dataframe into a dataframe that can be feed into my market simulator to evaluate how the portfolio of that strategy performs.

My Q-Learner used %B and MACD to make the indicator states as explained above, while my manual strategy used enter and exit flags marked by the price relative to the Bollinger Bands and zero-line crossover of MACD, as explained in the manual strategy project. In order to test both strategies against benchmark, and see how each does comparatively, I ran this experiment with no impact value and 0-dollar commission. I had a starting cash value of $100000 and each strategy can hold maximums of: 1000 shares long, 1000 shares short, or 0 shares. I am using no commission because I am assuming that having a value for either strategy would affect both strategies the same and therefore, it's not necessary for comparision. The following graph shows my output for this in-sample period and compares the performance of my Q-Learner (red line), manual strategy (black line), and benchmark (blue line) (of investing long on the first day):




This table also numerically compares each of the three strategies:

| | Strategy Learner | Manual Strategy | Benchmark |
|---|---|---|---|
| Sharpe Ratio | 1.4561943068 | 0.55574879545 | 0.304803642337 |
| Cumulative Return | 0.4247 | 0.1413 | 0.0661 |
| Final Portfolio Value | 142470.0 | 114130.0 | 106610.0 |

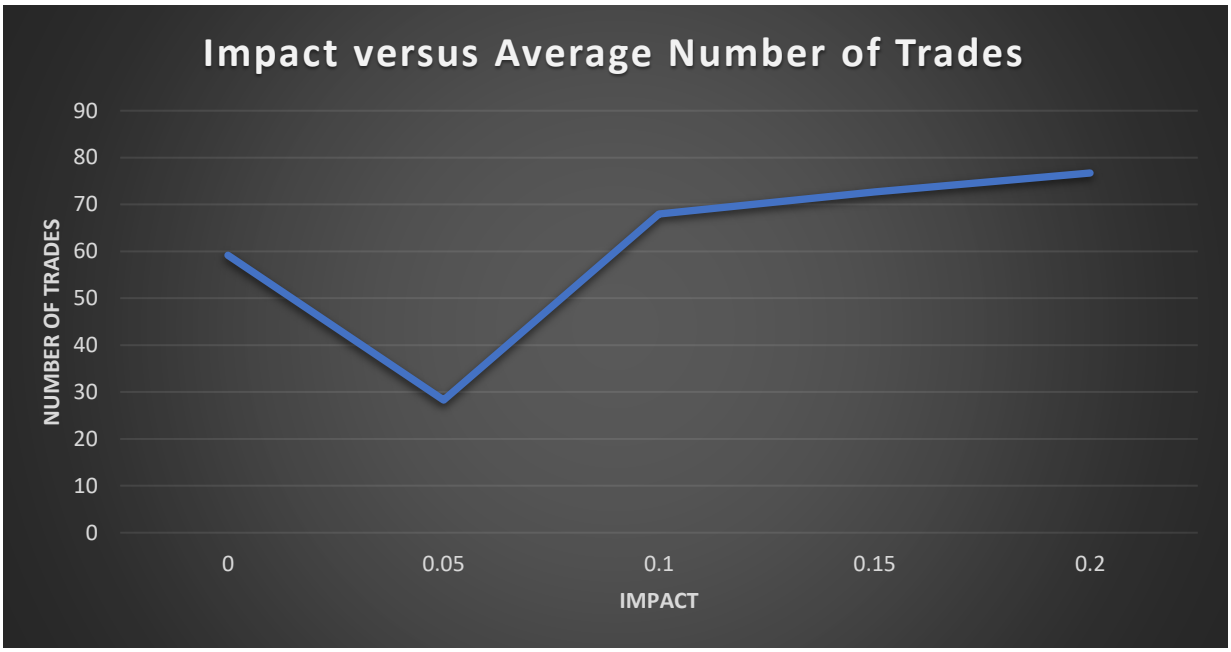
As can be seen from the graph and the table, my strategy learner outperforms the manual strategy and the benchmark considerably and even has higher reward to risk ratio (Sharpe ratio). I would expect this result, with the strategy learner performance beating manual strategy learner and benchmark performance, every time for in-sample data, as the Q-Learner will learn when it's best to make a certain trade because it can re-evaluate what it previously thought was the best action, from a certain state, through the process of a Q-update.

Experiment 2:

I believe that increasing the impact for the strategy learner and computing portfolio values will decrease the number of trades, decrease the Sharpe ratio, and decrease the final portfolio value /cumulative return of a portfolio. In order to test this hypothesis, I trained my Q-Learner on the in-sample period, using NFLX from January 1st 2003 to December 31st 2004 and the same indicators as explained above (BB% and MACD) with a minimum epoch count of 10 for the learning phase. I varied the impact from 0 to .2 in increments of 0.05, and trained the Q-Learner 10 times by calling addEvidence() 10 times for each impact. After training, I called testPolicy for each addEvidence on the same in-sample period and computed the number of trades made within the data-frame returned by testPolicy and the resulting portfolio values. In summary, I ran 10 trials of each impact value, and in each trial, I trained a new Q-Learner, got the policy it returns for the in-sample period, and computed number of trades and portfolio values for the policy based-trade that comes from each trial. The following chart showcases my results:

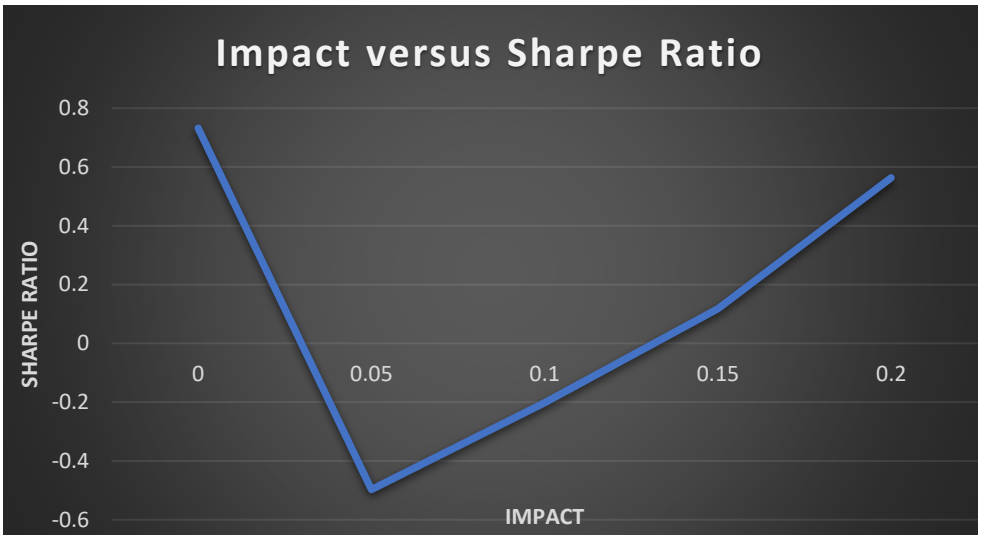
| | Average Number of Trades | Average Sharpe Ratio | Average Cumulative Return |
|---|---|---|---|
| **0** | 59.14286 | 0.732126 | 0.198171 |
| **0.05** | 28.28571 | -0.49742 | -0.4434 |
| **0.1** | 68 | -0.20216 | -2.33982 |
| **0.15** | 72.71429 | 0.117528 | -3.86377 |
| **0.2** | 76.71429 | 0.563136 | -5.39096 |

My results show that there seems to be a positive correlation between impact and number of trades. Except for impact=0.05, which seemed to have an outlier trial bringing down the average, the general trend seems that as the impact increases, the number of trades went up, according to my results, as shown here in a graph:



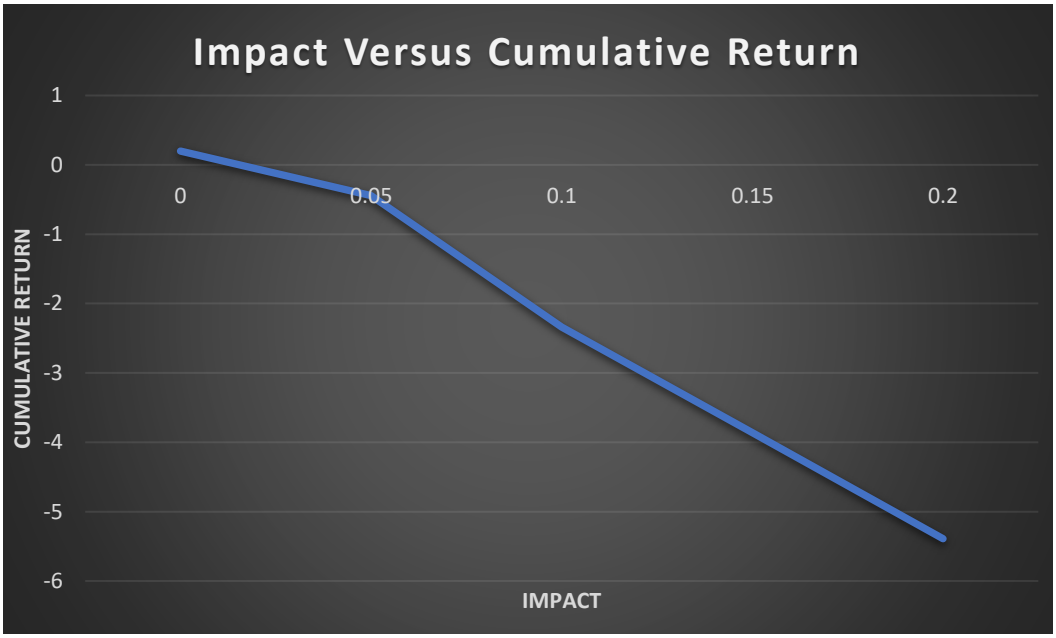

I am unsure why the number of trades goes up as the impact rises. It would seem to me that the number of trades should go down because with a bigger impact, you want to wait for higher profits to offset the impact that counts against you, before making a trade, which would mean there would be less overall trades. For example, if there was a low impact, say 0, then even a minimal rise in stock price seems to be profitable and something we would want to take advantage of. But a higher impact, say .25, would make that profit not worth because what I would be gaining per stock is offset by the impact and thus, I would theoretically wait for higher profits. Perhaps, I need to rerun this test in the future, with more time, or try to find out what exactly causes this as this is a very interesting trend.

My results show that there seems to be no linear correlation between Sharpe ratio and impact. Impact seems to not affect reward to risk ratio as shown in my graph:




There seems to be no correlation as impact rises because the sharpe ratio falls and rises between both ends of the impact.

My results show there seems to be a negative linear correlation between impact and cumulative returns. The general trend seems to be that as impact goes up, the average cumulative return goes down, as shown in a graph here:




This trend makes sense since a higher impact means we want to wait for higher returns before we enter or exit positions to offset the impact which counts against us. Thus, we cannot make profit off the accumulation of smaller profits that we might realize as lower levels of impact that we would not able to at higher levels. Thus, as impact increases, the cumulative returns of the portfolio decrease.