Project 8

Alonso Torres-Hotrum
atorreshotrum3@gatech.edu

Overview

For this project, a learning trading agent was implemented using a reinforcement learning based approach. The agent was trained using Q-learning for multiple episodes over a specified training period until it converged to a solution. Then, with learning turned off, the agent was tested using out of sample data and the cumulative return was evaluated. The agent makes market order decisions based on three market indicators: BBP, RSI, and MACD.

The data from each indicator was discretized by mapping it to recommendations, with 0 meaning hold, 1 meaning short, and 2 meaning long. Table 1 shows the value thresholds for the recommendations of each indicator. These recommendation thresholds are the same as those used in the manual strategy for Project 6. Note that MACD Difference refers to the difference between the MACD and Signal Lines using 9 day Exponential Moving Averages.

Table 1. A chart showing the recommendation thresholds for each indicator used, where x is the indicator value at a given time.

	Hold (0)	Short (1)	Long (2)
BBP	$0 \leq x \leq 100$	$x > 100$	$x < 0$
RSI	$30 \leq x \leq 70$	$x > 70$	$x < 30$
MACD Difference	No crossing	Crosses from positive to negative	Crosses from negative to positive.

The states used by the Q-Learner were each represented by an array consisting of four values, the first three of which corresponded to the recommendations of the three indicators. The last value of the array was the current position, which had the same possible values and corresponding meanings as the indicator recommendations. Figure 1 shows an example of how a state was represented in the array. Note that because each state was made up of a combination of four elements and each element had three possible values, the total number of possible states was $3^4 = 81$.

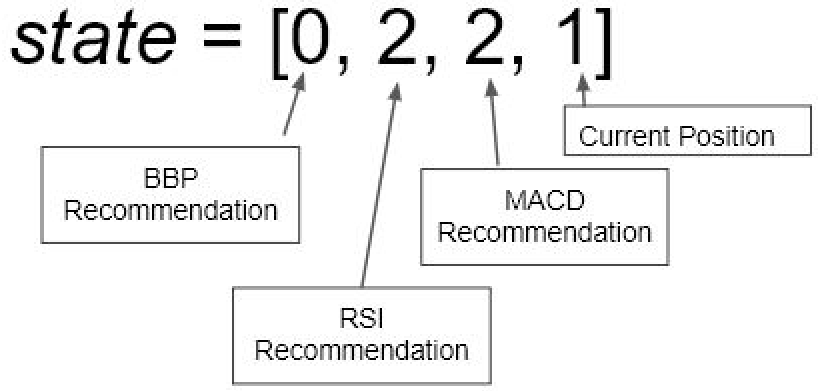

Figure 1. A breakdown of the four elements that constitute a state. Note that 0 means hold, 1 means short, and 2 means long.

Several different reward systems were considered and tested for this project. The learner was initially set to receive a small reward at each step if it correctly predicted how the market would change the next day, and a large reward if its final cumulative reward beat that of the benchmark. However, better results were realized by switching to a reward system that at each step gave a positive or negative reward proportional to the market return seen the next day and gave no final reward for overall performance. Impact was implemented in the reward system by having positive rewards reduced and negative rewards enhanced by the impact percentage.

The overall method the agent uses for learning market strategies is as follows:

- Initialize the impact, which is assumed to be zero if not specified.
- Initialize in-sample and out of sample dates as specified.
- addEvidence function:
    - Initialize the symbol, in-sample start date, in-sample end date, and starting cash amount.
    - Get the adjusted close prices for symbol throughout all in-sample trading days.
    - Get the daily returns for each day in the in-sample date range.

- Compute the benchmark performance:
    - Use the compute_portvals function from marketsimcode.py.
        - Pass in the symbol and in-sample date range.
        - Assume that 1000 shares are bought on the first day, and the position is held until the last day, when the 1000 shares are sold.
    - Compute the benchmark cumulative return.
- Compute the technical indicators for each day in the in-sample date range using the b_band, RSI, and MACD functions from indicators.py.
    - Place in an indicators dataframe.
- Instantiate the Q Learner from QLearner.py.
    - 81 total possible states, 3 possible actions.
    - alpha = 0.2, gamma = 0.9, rar = 0.5, radr = 0.99
    - No dyna used.
- Repeat until Q table converges:
    - Initialize df_trades dataframe.
    - Initialize holdings to zero.
    - Get current state *s*.
    - Use *querysetstate* function from QLearner.py to get initial action *a*.
    - Calculate the reward for initial action.
    - Perform the action and update df_trades.
    - Calculate next state *s'*.
    - For each day in the in-sample date range (excluding the first day):
        - Use the *query* function from QLearner.py to get next action.
        - Perform the action and update df_trades.
        - Get the daily reward for the action.
        - Calculate the new state.
    - Compute the learner's performance:
        - Use the compute_portvals function from marketsimcode.py.

                - Pass in the symbol and in-sample date range.
                - Pass in df_trades as the order sheet.
            - Compute the learner's cumulative return.
        - Update convergence criteria
            - Minimum number of episodes is 5, maximum is 20.
            - If the cumulative reward is greater than 1.5 and is within 0.001 of the previous cumulative reward, the table has converged.
            - If the cumulative return equals zero, reinstantiate the Q Learner and start over.
                - Only allowed once per iteration.
            - If after 20 episodes the cumulative reward is less than 1.5, reinstantiate the Q Learner and start over.
                - Only allowed once per iteration.
- testPolicy function:
    - Initialize the symbol, start date, end date, and starting cash amount.
    - Initialize df_trades dataframe.
    - Initialize holdings to zero.
    - Get current state *s*.
    - Use *querysetstate* function from previously instantiated QLearner to get initial action *a*.
    - Perform the action and update df_trades.
    - Calculate next state *s'*.
    - For each day in the specified date range (not including first day):
        - Use the *querystate* function from QLearner.py to get next action.
        - Perform the action and update df_trades.
        - Calculate the new state.
    - return df_trades

The Q table was initially considered to have converged if the cumulative return was greater than the benchmark cumulative return. In order to incentivize the learner to converge onto a higher cumulative return, the convergence criteria was changed to converge only if the cumulative return was greater than 1.5. If the learner converged to an answer less than 1.5, it instantiated a new Q Learner and started over. For runtime purposes, the learner was only allowed to restart once per episode.

The Q-Learner was always initialized with a mid-level exploration rate of 0.5 as it provided a balanced exploration/exploitation ratio and gave the best results. The agent performed sub-optimally in cases where it did not receive any positive rewards within the first three iterations, which makes sense as it never learned that it could obtain positive rewards. These cases would result in the learner repeatedly returning empty dataframes. A check was put in place to instantiate a new learner any time the previous learner began to return empty dataframes, after which the learner beat the benchmark 98 of 100 times when tested using JPM in-sample data.

Experiment 1

In this experiment, the manual strategy was compared against the Q-Learning strategy using in-sample testing (January 1, 2008 to December 31, 2009) for JPM. In order to calculate the theoretically optimal solution, it was assumed that all trades would be executed at the adjusted close value of that day. It was also assumed that there would be no slippage between the adjusted close value and the order execution value, and thus the impact was set to 0 for both strategies. No commissions were charged on trades and it was assumed that trades could execute on any day the market was open.

The Q-Learner was always initialized with alpha = 0.2, gamma = 0.9, rar = 0.5, radr = 0.99, and no dyna. It was assumed that the learner would have access to all in-sample data and could run until convergence was reached. The Q-Learner and the manual strategy both operated using the same market indicators as were used in Project 6: BBP, RSI, and MACD. The benchmark strategy assumed buying 1000 shares of JPM on the first day of the in-sample data, holding, then selling all of the shares on the last day of the in-sample data.

The experiment consisted of using the QLearner class' *addEvidence* method to repeatedly train the learner over the training data until convergence, then querying it to obtain a final order sheet using the QLearner class' *testPolicy* method. The manual strategy was executed using the *testPolicy* method in the ManualStrategy class to obtain its corresponding order sheet. Finally, the benchmark order sheet was built to reflect its two order strategy.

Each order sheet was then run through the *compute_portvals* function from marketsimcode.py in order to get their corresponding values for each day in the in-sample date range. The final step was to normalize and plot the learner, manual, and benchmark portfolios along with the value of JPM over the in-sample dates. The resulting graph can be seen in Figure 2.

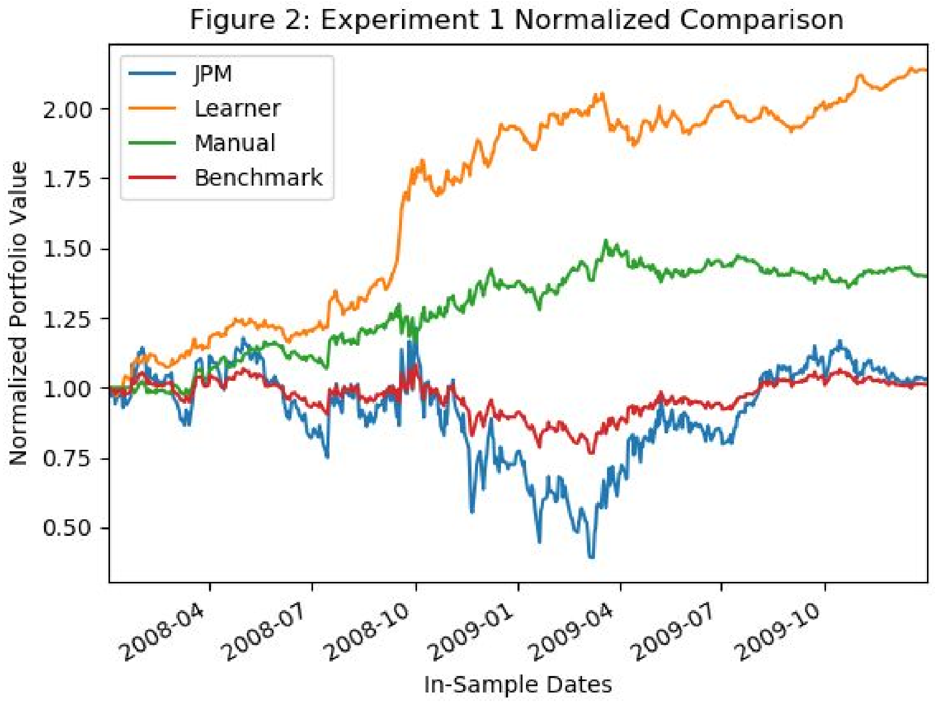

Figure 2. The results of Experiment 1.

JPM's stock price varied significantly throughout the in-sample dates, and ended with a value 3.2% higher than its original value. The overall cumulative returns were 2.063 for the Q-Learner, 0.400 for the manual strategy, 0.012 for the benchmark. Thus, the learner achieved significantly higher results than the manual or benchmark strategies.

It was not always the case that the Q-learner beat the other strategies. Due to its reliance on randomly generated numbers, the learner's cumulative returns consistently changed and ranged from 0.29 to 2.86 when the experiment was repeated 100 times. With that said, the average performance of the Q-Learner over those 100 tests was 44.3% better than that of the manual strategy and 4596.5%

better than that of the benchmark. Thus, while the learner's relative results may not have always beat those of the manual and benchmark strategies, its overall in-sample performance was superior to both.

Experiment 2

In Experiment 2 the effect of the *impact* variable was analyzed by varying its value and noting how it affected the cumulative reward. Once again the experiment was conducted using JPM on the in-sample period. The learners that were generated for this experiment used the same indicators and had the same parameters as those in Experiment 1.

My hypothesis is that as the impact increases the cumulative rewards will steadily decrease and the volatility will increase. This will be due to the impact's effect on the reward system, which will affect the learner's ability to learn from its actions. I further hypothesize that at some point the impact will be so large that the learner will begin to return empty order sheets, and the cumulative rewards and volatility will be zero from that point on.

The experiment methodology was to start with an impact of 0, then used the *addEvidence* and *testPolicy* methods of the StrategyLearner class to develop a Q-Learner and calculate its cumulative rewards. The impact was then incremented by 0.01 and the cumulative rewards were stored in a dataframe. The process was repeated 10 times, after which the resulting dataframe was plotted (Figure 3). The rolling mean and rolling standard deviation were then calculated using a window of 3 and placed in a dataframe, which was then plotted and can be seen in Figure 4.

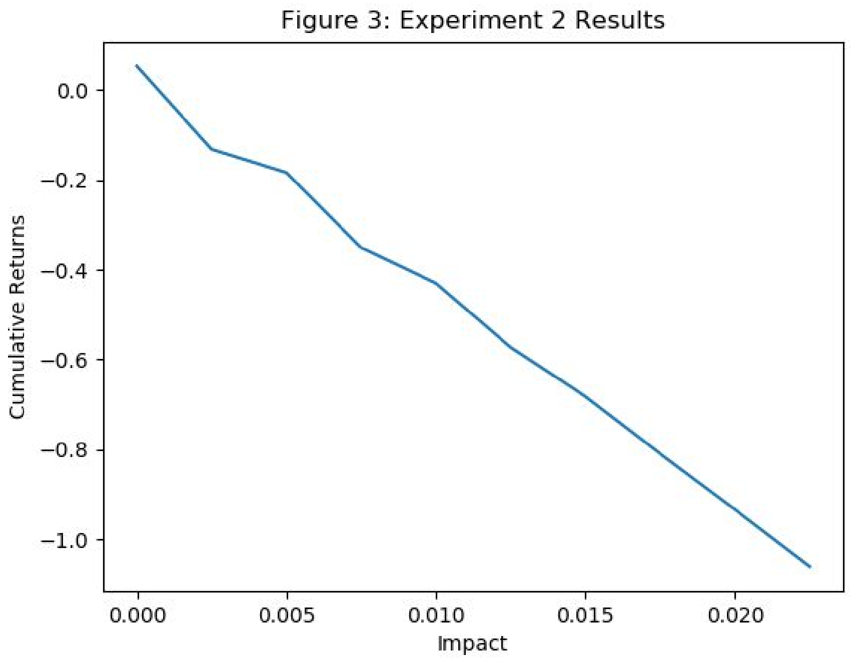

Figure 3. The results of Experiment 2.

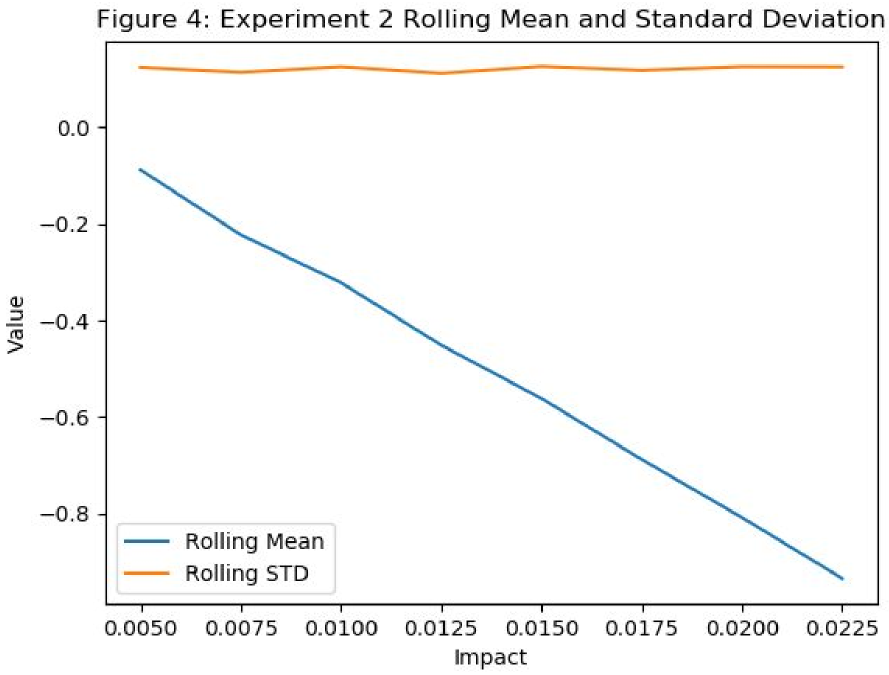

Figure 4. The rolling mean and standard deviation of Experiment 2.

My hypothesis was partially correct in that the cumulative returns did steadily decrease as the impact increased. However, the volatility did not increase as predicted. Likewise, the cumulative return did not eventually go jump to zero as I had hypothesized. It is possible that these occurrences do take place for higher impact values, but for the range tested the results clearly indicate that the cumulative return steadily and consistently decreased as the impact increased.

In order to fully understand the effect that impact has on the Q-Learner a wider range of impact values should be explored. This would allow for a more complete analysis of cumulative return and volatility trends as they relate to impact. Unfortunately, due to the time required to train learners and the runtime restrictions of this project, the number of impact values tested was limited to ten.