STRATEGY LEARNER

Allan Reyes, reyallan@gatech.edu

INTRODUCTION

Automated trading has always been of interest to investors, and with the recent advances in computer hardware and the availability of big data, they have started to turn to the area of Machine Learning in hopes of producing trading models that produce positive returns. Several techniques can be utilized to learn such models; one of which is Q-Learning, a Reinforcement Learning algorithm. In this report, a description of how this algorithm was applied to trading, the experiments performed to assess its performance, and their results, are presented.

TRADING AS A REINFORCEMENT LEARNING TASK

To apply Q-Learning to a particular problem, it needs to be correctly framed as a Reinforcement Learning task. All such tasks have states, actions, and rewards. States describe all the necessary information about the world while actions determine the way the RL agent can interact with the environment. On the other hand, the rewards determine the positive or negative reinforcement that the learner uses to flag its actions as *good* or *bad*, and thus derive the most optimal behavior. The following sections present an explanation of how each of these elements was applied to trading.

STATES

In the trading world, the states are a combination of different technical indicators applied to a particular stock. These indicators describe core aspects about a stock, like its price movement or whether is regressing to the mean, making them the perfect representation of the trading environment. Now, it is important to note that, for this project, a *tabular* version of Q-Learning was implemented. This means that the number of states must be **finite** in order to be represented as a table of Q-values. For this reason, all indicators were **discretized**. The state was then also combined into a single number using a combination of said discretization. Following is an explanation of the technical indicators that were used for this project along with their discretization technique.

MOMENTUM

Momentum is an indicator that describes how much the price has changed over a certain period of time. In other words, it gives the investor an idea of how the price has been oscillating and whether it is trending up or down. Momentum can be computed as follows:

1. Define a lookback period ***n***
2. Obtain all the historical ***prices*** for the stock over a certain period of time
3. Calculate the **momentum** for each trading day: *momentum[t] = (prices [t] / prices [t – n]) – 1*

Momentum is of interest because it provides a raw estimation of how strong the price of a stock is moving. In a trading strategy, this can signal BUY opportunities when the momentum is positive since the investor can expect, roughly, that the price will continue to increase. A similar reasoning signals SELL opportunities when the momentum is negative.

DISCRETIZATION ALGORITHM

Momentum typically ranges from -0.5 to 0.5, and so it was discretized into five different bins:

- **0:** x < -0.5
- **1:** -0.5 <= x <= 0.0
- **2:** 0.0 < x <= 0.5
- **3:** x > 0.5
- **4:** x == NaN

PRICE/SMA RATIO

Simple Moving Average (SMA) can be defined as the *rolling* average over a window of time of the price of a stock. In other words, it describes the mean of the price over a lookback period. Now, price/SMA ratio is a technical indicator that measures how much the price has diverged from its rolling average, and it can be computed as follows:

1. Define a lookback period ***n***
2. Obtain all the historical ***prices*** for the stock over a certain period of time
3. Calculate the **price/SMA** for each trading day: *ratio[t] = (prices[t] / prices[t-n:t].mean()) – 1*

The Simple Moving Average serves as an estimation of the *true* value of a stock so, in a trading strategy, an investor can look for instances where the price of a stock crosses the SMA that can signal particular trading opportunities. In particular, situations where the price went above the SMA and down which suggests the price is regressing back to the mean, and vice versa; this is what Price/SMA ratio is precisely measuring.

DISCRETIZATION ALGORITHM

Simple Moving Average also typically ranges from -0.5 to 0.5, so the same discretization technique was used as with Momentum.

BOLLINGER BANDS®

Bollinger Bands® (BBands) are, as the name implies, *bands* two standard deviations above and beyond the Simple Moving Average (SMA). For this project, the BBands values were used as a technical indicator to measure the difference between the SMA and the price with respect to two standard deviations of the mean. The BBands values can be computed as follows:

1. Define a lookback period ***n***
2. Obtain all the historical ***prices*** for the stock over a certain period of time
3. Calculate the ***SMA,*** as previously described
4. Calculate the **BBands** for each trading day: *bbands[t] = (prices[t] - sma[t]) / (2 * sma[t].std()*

In a trading strategy, the Bollinger Bands® values serve as an indication of whether the stock appears to be **oversold** (the value seems to be underrated) or **overbought** (the value seems to be overrated). In terms of the BBands values, a stock can be flagged as oversold if the price goes below the bottom band and then crosses up; and flagged as overbought if the price goes up the top band and then crosses back down.

DISCRETIZATION ALGORITHM

Bollinger Bands® typically ranges from -1.0 to 1.0 so five different bins were used for discretization:

- **0:** x < -1.0
- **1:** -1.0 <= x <= 0.0
- **2:** 0.0 < x <= 1.0
- **3:** x > 1.0
- **4:** x == NaN

MONEY FLOW INDEX

The Money Flow Index (MFI)[1] is a technical indicator that measures the flow of incoming and outcoming money over a particular period of time. In other words, the MFI uses the price and volume of a stock to measure buy and sell pressure to produce the index. The MFI is computed as follows:

1. Define a lookback period ***n***
2. Obtain all historical ***high, low*** and ***closing prices***, as well as, the ***volume*** for the stock
3. Calculate the ***typical price*** as: *typical[t] = (high[t] + low[t] + close[t]) / 3*
4. Calculate the ***raw money flow*** as: *raw[t] = typical[t] * volume[t]*
5. Calculate the ***n-day positive money*** by summing up all of the raw money flow on days where the typical price is higher than the day before
6. Calculate the ***n-day negative money*** by summing up all of the raw money flow on days where the typical price is lower than the day before
7. Calculate the ***money flow ratio*** as: *ratio[t] = positive[t] / negative[t]*
8. Calculate the ***MFI*** as: *mfi[t] = 100 – (100 / (1 + ratio[t]))*

[1] http://www.investopedia.com/terms/m/mfi.asp

Money Flow Index is used in a trading strategy as leading indication of **oversold** or **overbought** instances for a particular stock. According to the creators of this index, Gene Quong and Avrum Soudack, an MFI of over 70 suggests the stock is overbought, while a value lower than 30 suggests that it is oversold.

DISCRETIZATION ALGORITHM

The typical range for Money Flow Index is from 0 to 100, and since MFI has very specific boundaries, these were used to discretize the values into four bins:

- **0:** x < 30 (oversold bucket)
- **1:** 30 <= x <= 70 (no indication)
- **2:** x > 70 (overbought bucket)
- **3:** x == NaN

ACTIONS

In terms of actions that the Q-Learning agent could take, the three following were used: **LONG**, **CASH** and **SHORT**. Where **LONG** meant buying to hold a +1000 position for a particular stock and **SHORT** meant selling to hold a -1000 position. As for **CASH**, the agent would buy or sell the appropriate number of shares to close any current long or short positions.

REWARDS

The reward was computed as the **daily return** of the current position. Every training iteration the stock price was compared with that of the previous iteration along with the position the agent had. For example, if the agent were holding a long position and the price goes up, then it would receive a **positive** reward since the daily return would also be positive. On the other hand, if the price actually goes down, then it would receive a **negative** reward to reinforce the fact that a long position was not the best decision. This same methodology applies to short positions.

MARKET IMPACT

A very important aspect of how the rewards were computed is the introduction of **market impact.** Market impact is used to account for unexpected changes in the market, which for the purposes of learning trading strategies, it should always go **against** the learner's own expectations. Impact was used to affect the current price of the stock in different ways depending on what the current position was. If the agent decided to go long, the current price was **reduced** by the market impact effectively simulating an unexpected drop of the price that the agent did not account for. Similarly, for short positions, the price was **increased** by the impact.

EXPERIMENTATION

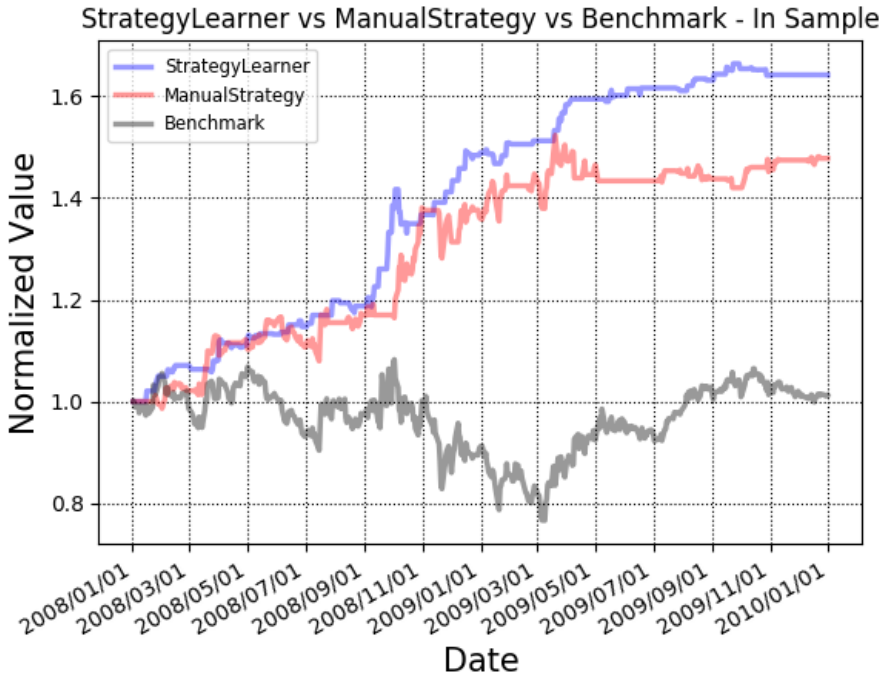
Two experiments were performed to evaluate different aspects of the strategy learner developed for this project. The following table describes the parameters that were used to instantiate the Q-Learning trading agent for both experiments.

Parameter	Value	Description
Number of states	4,444	Largest number that can be created based on the discretization buckets
Number of actions	3	LONG, CASH or SHORT
Alpha	0.2	The learning rate
Gamma	0.9	The discount factor
Epsilon	0.5	The random action rate
Epsilon Decay	0.99	The decay factor for the random action rate

The following sections describe each of the two experiments including specific parameters, assumptions and their results.

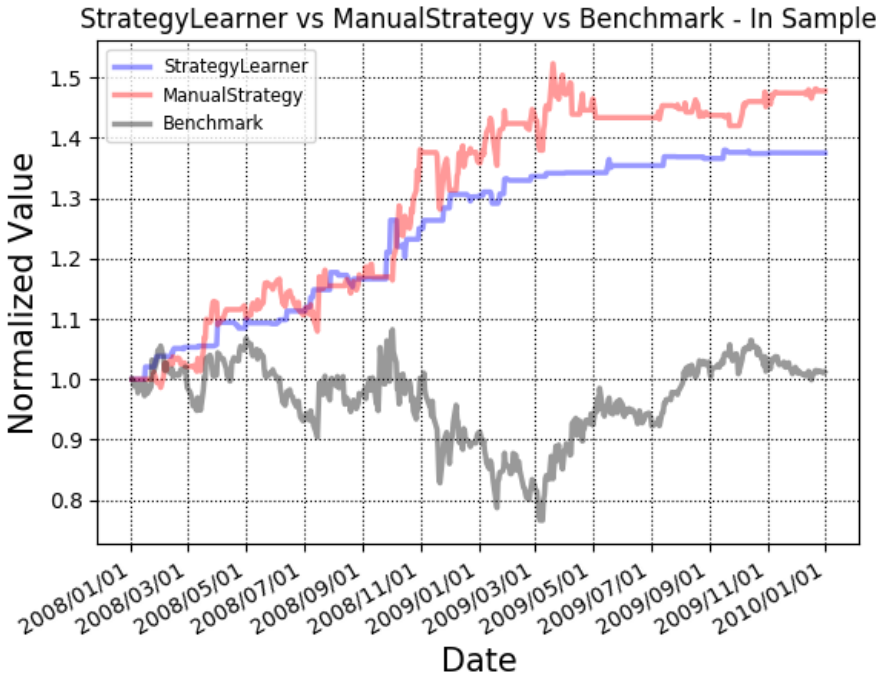
EXPERIMENT 1: PERFORMANCE

For the first experiment, the in-sample performance of the strategy learner was matched with the performance of a rule-based manual strategy and a specific benchmark which consisted of buying 1000 shares of the stock and holding it until the end of the period. In terms of parameters, the stock symbol that was used was JPM, the in-sample period went from January 1st, 2008 to December 31st, 2009 and the starting cash was $100,000. For this particular experiment, commission was assumed to be $0.00, and market impact was not considered either (i.e. a value of 0 was used). Finally, the value 1481090000 was used as a seed for the underlying random number generators. This was done to allow for reproducibility of the results shown here. The following graph shows the normalized values for each of the strategies.



As it can be seen from the figure above, the Q-Learning agent was able to produce a strategy that yielded better results than both the manual strategy and the benchmark. In particular, it obtained a cumulative return of 64% in contrast to the 48% of the manual strategy: a 1.3x improvement!

Though these results are very promising, it is crucial to acknowledge one caveat: the random generator was fixed using a particular seed. But, does this mean that this relative result should not be expected every time with the in-simple data? From further experimentation, it seems that this result is quite consistent irrespective of the seed used. Out of a couple dozen more trials that were run, each without setting a particular seed, only one such trial resulted in a strategy that did not beat the manual one. The following figure shows the results from that particular run.



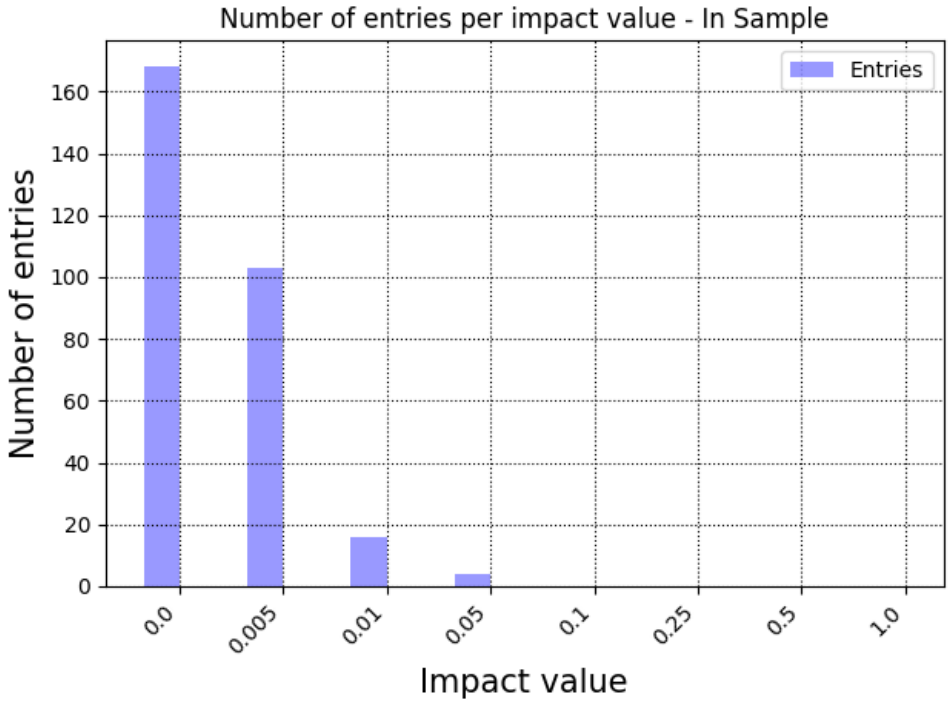
This comes from the fact that Q-Learning is sensitive to exploration and exploitation of the policy. In other words, if the underlying random number generator consistently returns values that motivate exploitation, the learning agent might converge into sub-optimal policies because not all possible were explored. This issue can be addressed by using different exploration strategies which was out-of-scope for this project.

EXPERIMENT 2: EFFECT OF MARKET IMPACT

Market impact is an important aspect that not only needs to be taken into account when simulating transactions, but also when implementing agents that learn strategies. In this second experiment, the effects of the market impact were studied in order to validate the following hypothesis: *the impact encapsulates the overall fluctuation of the market; in particular, movements that would affect one's portfolio. For the case of the strategy learner, impact should directly affect the learned policy in terms of the willingness of the agent to take risks by betting on the behavior of the market.*

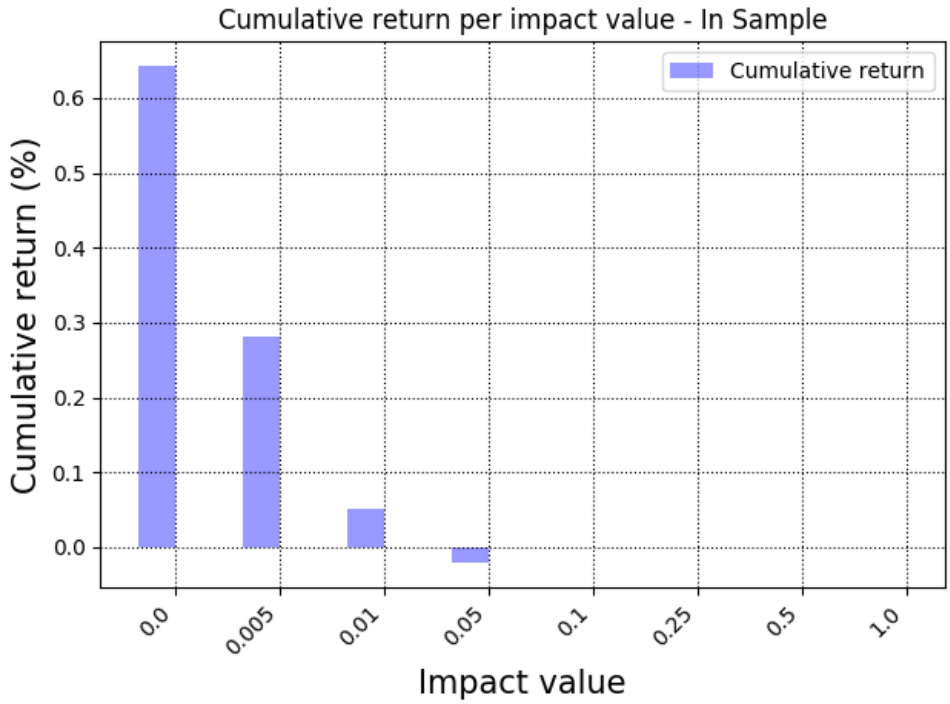
To evaluate this hypothesis two metrics were used: **number of entries** and **cumulative return** of the strategy. These metrics were computed by training and querying different strategy learners with various

values of market impact: 0.0, 0.005, 0.01, 0.05, 0.1, 0.25, 0.5 and 1.0. All learners used JPM as the stock symbol, the in-sample period, a starting value of $100,000 and $0.00 in commissions. Finally, a value of 1481090000 was used as the seed to allow reproducibility of all the results. The following graph shows the **number of entries** (either long or short) for each market value.

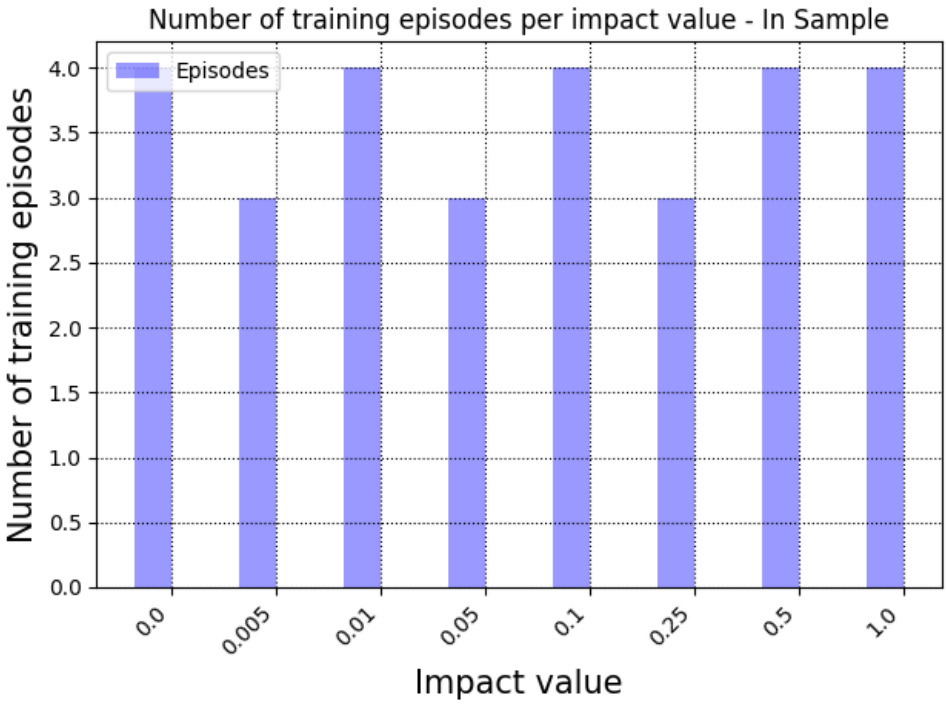


Notice how the number of entries gets reduced as the market impact increases. This metric shows the strategy learner being more cautious about holding specific positions. In the extreme, where the impact of the market is large, the learning agent is so uncertain about how the price will fluctuate that the optimal policy is to not buy or sell any shares! In contrast, when the market impact is low, performing trades is much safer. These results validate the hypothesis in terms of how the impact is affecting the learned policy.

Directly related to the point mentioned above, as the market impact increases and the agent's desire to take risks decreases, so is the **overall performance** of the strategy until it reaches a point where no return is observed because no trades where issued. These results can be seen in the following figure.



Finally, one last metric was computed but it was done for the author's mere curiosity. It was interesting to see how as the market impact increases; the number of complete learning episodes is not affected, as seen in the following graph. The impact does not affect the *rate* of convergence, but rather the *strategy* the agent converges to!



CONCLUSION

In this project, a Q-Learner was implemented by framing the trading environment as a Reinforcement Learning task where discretized indicators were used as states, trading decisions as actions and daily returns as rewards. The performance was evaluated against a rule-based strategy and a benchmark where results showed it was able to beat both the other strategies. The effect of the market impact was also evaluated to validate the hypothesis that impact directly affects the willingness of the agent to execute trades. Metrics like number of entries and cumulative return per market value were computed and confirmed this effect. Finally, this project demonstrated that Q-Learning can be applied to trading with positive results; however, market impact should be taken into account to optimize hyper-parameters to devise strategies that are more resilient and robust.