**Report – Machine Learning for Trading**

**Project MC3P4 – Trading Strategy Learner**

-Submitted by Yashovardhan Jallan

## ➢ DESCRIPTION OF MY STRATEGYLEARNER.PY

1) *_init_()* Function

   I am creating a constructor for the QLearner in this function. I have initialized it as
   *Self.learner=ql.QLearner()*

2) *addEvidence()* Function
   In this function, the training dataset is used to formulate indicators which is then used to train the Q-Learner.

   i. First, the prices of the stock, let's say IBM, is collected in a dataframe named '*prices'*. The util.py file is used to get this.

   ii. Choice of Indicators: I have used 3 technical indicators
      a) Bollinger Bands
      b) Momentum
      c) Volatility

   iii. Now since Bollinger Bands uses rolling Standard Deviation values and Simple Moving Average, I decided to collect stock price data for 30 days before start date of the training as well so that I have a value of these indicators on the 1st day as well.

   iv. Binning: Now since the indicators give us a continuous value, and our QLearner.py is coded in a way to accept discrete states, I decided to discretize the value of the indicators to put them into Bins.

   v. I have used a built-in function in pandas called 'qcut' to help me with the discretization. This function accepts a range of values and has the number of bins as a hyper parameter. Also, this function makes sure that each bin has equal number of observations in it.

   vi. Once the binning is done, I have combined the 3 indicators to create a unique state as Mr. Byrd suggested in class. I have chosen 10 bins for each indicator, so I get a total of 10^3 = 1000 combinations. So, I can have 1000 unique states. This is achieved by *State=100*(Label of Indicator1)+ 10*(Label of Indicator2)+1*(Label of Indicator3).*

   vii. Before I begin updating the Q-table, I have initialized a '*holdings'* dataframe with index as market open days, with columns tracking '#Shares', "Cash","Price of Stock", "Trades" and "Portfolio Value".

viii. Q-Learning Begins: Now, I have coded the for loop for QLearning with a changeable epoch value. I have used Daily returns on the portfolio as my reward function. The approach for Q-learning is:

*for loop of a certain number of epochs:*

*state=state on day 0*

*action=Qlearner.querysetstate(state)*

*according to the action received, update my Holdings dataframe*

*reward=(portval_day1/portval_day0)-1*

*s_prime=state on day 2*

*for loop with index i to run through all the days in the training dataset:*

*action= Qlearner.query(s_prime, reward)*

*according to the action received, update my Holdings dataframe making sure that the rules of the trading environment are not violated, i.e. at a time, my stock position should be anything other than +200, 0 or -200.*

*Reward= [ portval_day(i+1)/ portval_day(i) ] -1*

*S_prime = state on day(i+1)*

ix. The pseudo code above shows the main logic of my Q-learning code. I am using three action, +200, 0 or -200. These correspond to the stock position on a given day.

x. Updating the *holdings* dataframe: This is the dataframe which stores value according to the rules of the environment. My QLearner can give me three actions as I have mentioned above. So in total we have 9 possible cases. The table below shows the different scenarios which may arise.

| Position on Day 1 | Position on Day2 | Update |
|---|---|---|
| +200 | +200 | No update in Cash value as no trade made. |
| | 0 | Cash(day2)=Cash(day1)+200*stockprice |
| | -200 | Cash(day2)=Cash(day1)+400*stockprice |
| 0 | +200 | Cash(day2)=Cash(day1)-200*stockprice |
| | 0 | No update in Cash value as no trade made. |
| | -200 | Cash(day2)=Cash(day1)+200*stockprice |
| -200 | +200 | Cash(day2)=Cash(day1)-400*stockprice |
| | 0 | Cash(day2)=Cash(day1)-200*stockprice |
| | -200 | No update in Cash value as no trade made. |

xi. Reward Function: I have used the daily returns on my portfolio value as the reward function for QLearning.

Portfolio Value on a certain day = (Stock position*Stock Price) + Cash in Hand

Reward = [ Portfolio value on day(i+1) / Portfolio value on day(i) ] - 1

3) testPolicy() Function

i. Now once the training is done, we move on to the testPolicy() function. This is where we use the updated QTable to make predictions or give us a trading policy.

ii. The structure of this function is very similar to addEvidence() function. Based on the stock prices, we calculate the values of our Technical Indicators which are (a) Bollinger bands (b) Momentum and (c) Volatility.

iii. Then based on the values of the technical indicators, I have discretized them and created bins to feed into the QLearner.py

iv. I have created a '*holdings'* dataframe as before which keeps track of '#Shares', "Cash", "Price of Stock", "Trades" and "Portfolio Value".

v. Now, since the QLearner Table is already created, we do not need to update it any further. Hence, for each trading day, I have queried the QTable value using *querysetstate()* to get the suggested action.

*state=state on day 0*

*action=Qlearner.querysetstate(state)*

*according to the action received, update my Holdings dataframe*

*s_prime=state on day 2*

*for loop with index i to run through all the days in the testing dataset:*

*action= Qlearner.querysetstate(s_prime)*

*according to the action received, update my Holdings dataframe making sure that the rules of the trading environment are not violated, i.e. at a time, my stock position should be anything other than +200, 0 or -200.*

*S_prime = state on day(i+1)*

vi. Finally, when I have run through all the trading days, my '*holdings'* dataframe is ready and updated. I return the *['trades']* column as the required answer.

## ➢ EXPERIMENTATION WITH HYPERPARAMETERS

There are many hyperparameters in this project which can be played around with. The QLearner itself has parameters like alpha, gamma, random action rate and random action decay rate.

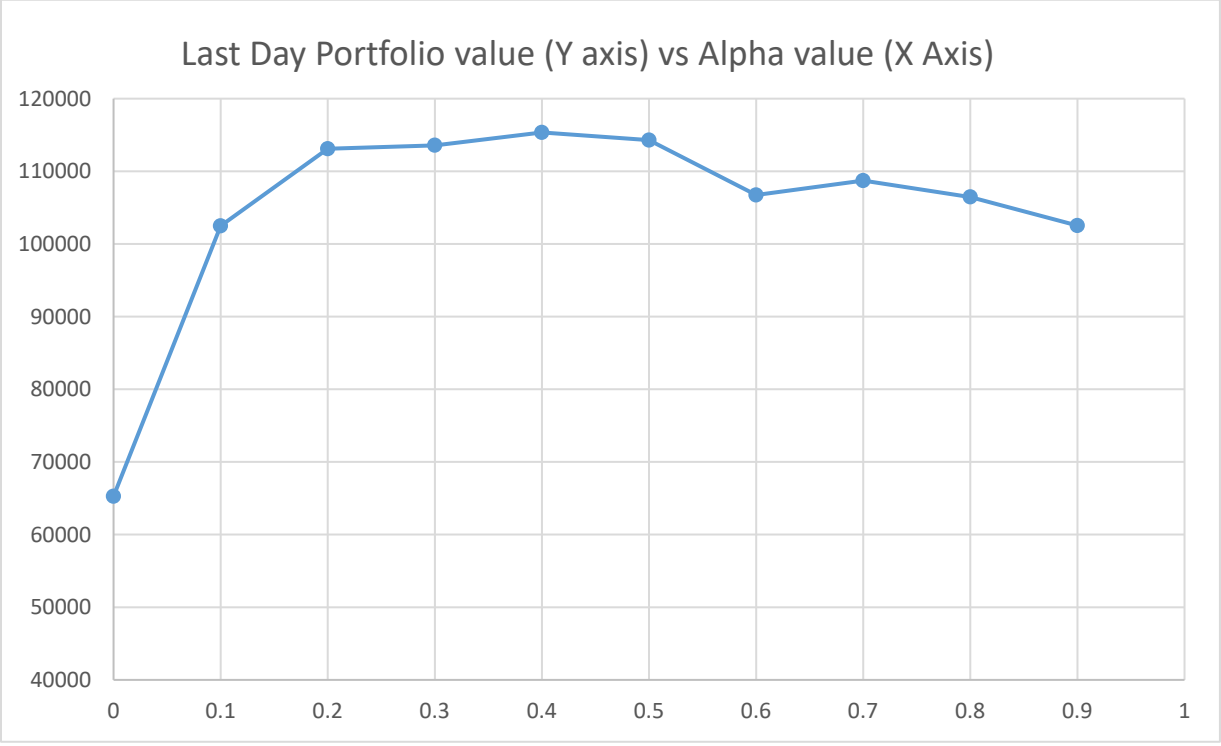
Other than the hyperparameters associated with the Q-learner, we have several of them associated with our technical indicators. I have used (a) Bollinger bands (b) Momentum and (c) Volatility. These make use of a Simple Moving Average (SMA) and Rolling Standard Deviation. So we can choose the number of days or the window size which we want to consider while calculating this parameter.

Also, the number of states for QLearning is also a hyperparameter. We can essentially choose the number of discrete states that we want to create for Qlearning.

I am running the experimentation on 'AAPL'. The training period is same as the grade_strategy_learner.py which is 1st Jan, 2008 to 31st Dec, 2009. The test period is 1st Jan, 2010 to 31st Dec, 2011. The starting value is $100,000. I am comparing the portfolio value at the end of the testing period for each experiment. The goal is to achieve a higher portfolio value.

I have chosen the following two hyperparameters. *Alpha (QLearning Parameter)* and *Window size to calculate Momentum*.

1. Alpha



As we can see from the graph above, as alpha is varied from 0 to 0.9, the last day portfolio value changes. An alpha value somewhere between 0.3 to 0.5, gives us the best returns.

2. Window size for Momentum

Varying the window size of momentum does not give a very consistent trend at least in the AAPL dataset. When the window size is 1 day, it gives a portfolio value greater than starting value. However, if the window size is 4 days, the portfolio value is close to $50,000. And then, if I move it back to 5 days, it gives me a positive return. Keeping the window size between 5 days and 9 days still yield positive return on starting value. However, if it is 10 days, we can see that it dips below the starting value.