Reinforcement Learning for Optimized trade execution

Lu Liu
J5299207

1. Introduction

Description of the problem:
Many research has been done regarding the use of reinforcement learning in optimizing trade execution. It has been shown in many hedge fund and research labs that this has indeed succeeded in producing consistent profit (for a certain period of time) .
In this project, I try to implement a simplified version to study the basic working of such a mechanism:
Train a Q-Learner using historical stock data to form a policy, then use the trained policy to predict trade actions for new stock data.
The goal is to beat the market (stock price).

Assumptions and rules:
Trading is limited to one stock
assume no transaction fees
Assume our trading actions itself does not affect stock price in anyway
start trading with long 100 shares
starting portfolio value =price of 100 shares
limit to trade 100 shares per day, can not double down.
Actions: 0 = BUY, 1 = SELL, 2 = NOTHING
Positions: 0 = CASH, 1 = SHORT, 2 = LONG
State used for the q-learner:
states= (Bollinger Band Value) + (volatility)*10+ (volume)*100+Position*1000
There are 2999 different states

2. Requirements

Modules used: numpy, pandas, pandas_datareader, matplotlib,os,datetime

3. Description of the Python program .

get_data.py
Using pandas_datareader module to retrieve historical stock data from yahoo finance

driver_program.py

Initialize a q learner
set the stock symbol and time period parameters for both training and testing
Kick start the learning process
Then test the learner

q_learner.py:
This is a q-learner class with two functions:
query_set_state : return the optimal action for the provided state according to the q-table
query_up_date: given the new state and reward obtained from executing the last action, update the q-table, and return an action to be executed for the next step.
Note: This q_learner program is independent of the problem we are solving, its state, action and reward parameters can be defined for different problems.

policy_learner.py:
The PolicyLearner class contain functions that train the learner and test for the learning result.

util.py:
Utility functions to read data and plot data.

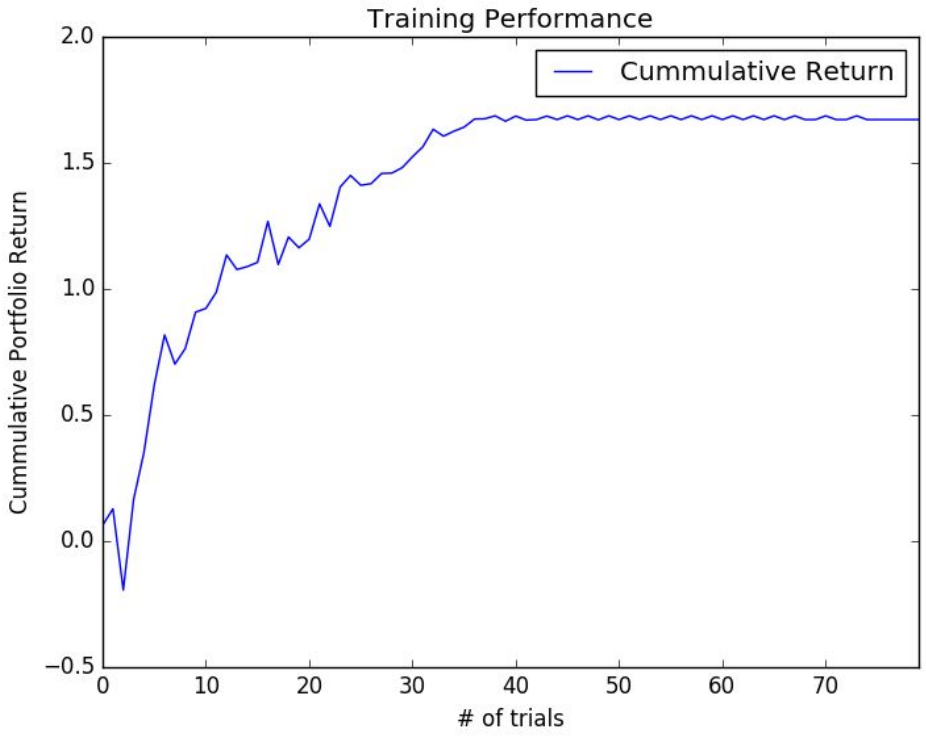
Python elements used :
1. Data structure
2. Functions
3. Classes
4. Importing external modules
5. Error checks using try-except
6. File input and output
7. Generator

4. Program output

The program was tested on five stocks: AAPL, IBM, GOOG, SPY , CELG
Training was done on three years of historical data, then tested the trained q-table to trade for three month.
Training was done for 80 trials (one trials= one loop through three years data) , as it is observed that under my parameter settings for the q-learner, 80 trials are enough for q-learner to converge.

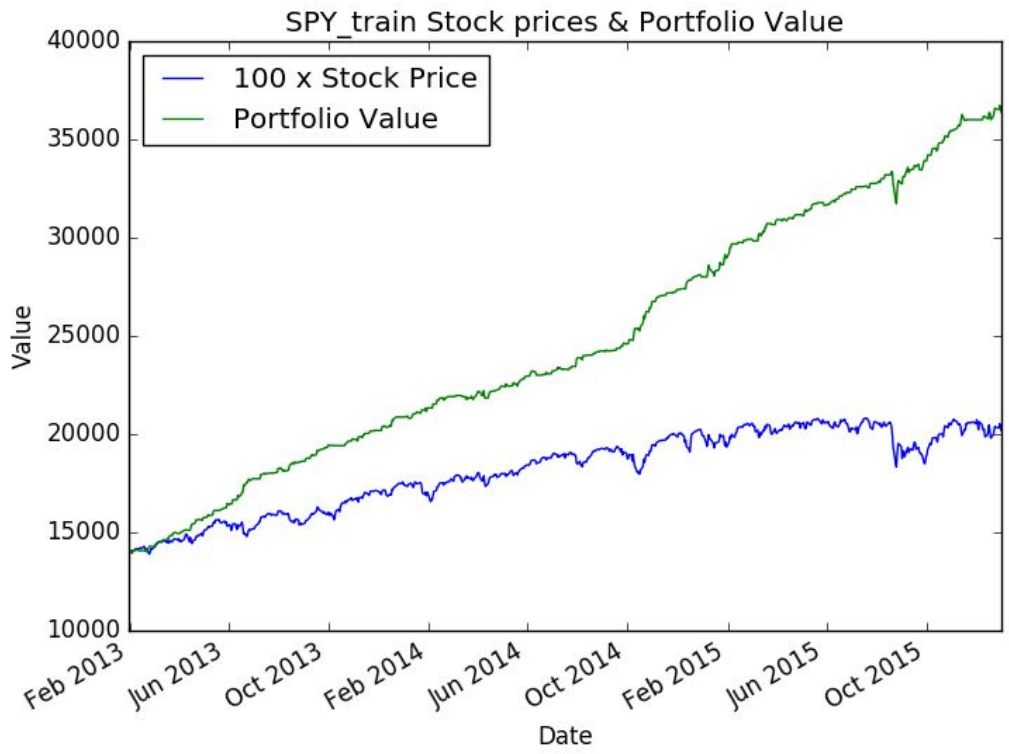
Description of result (using SPY as example, all results are enclosed in the zip file):

Figure 1: training performance vs # of trials



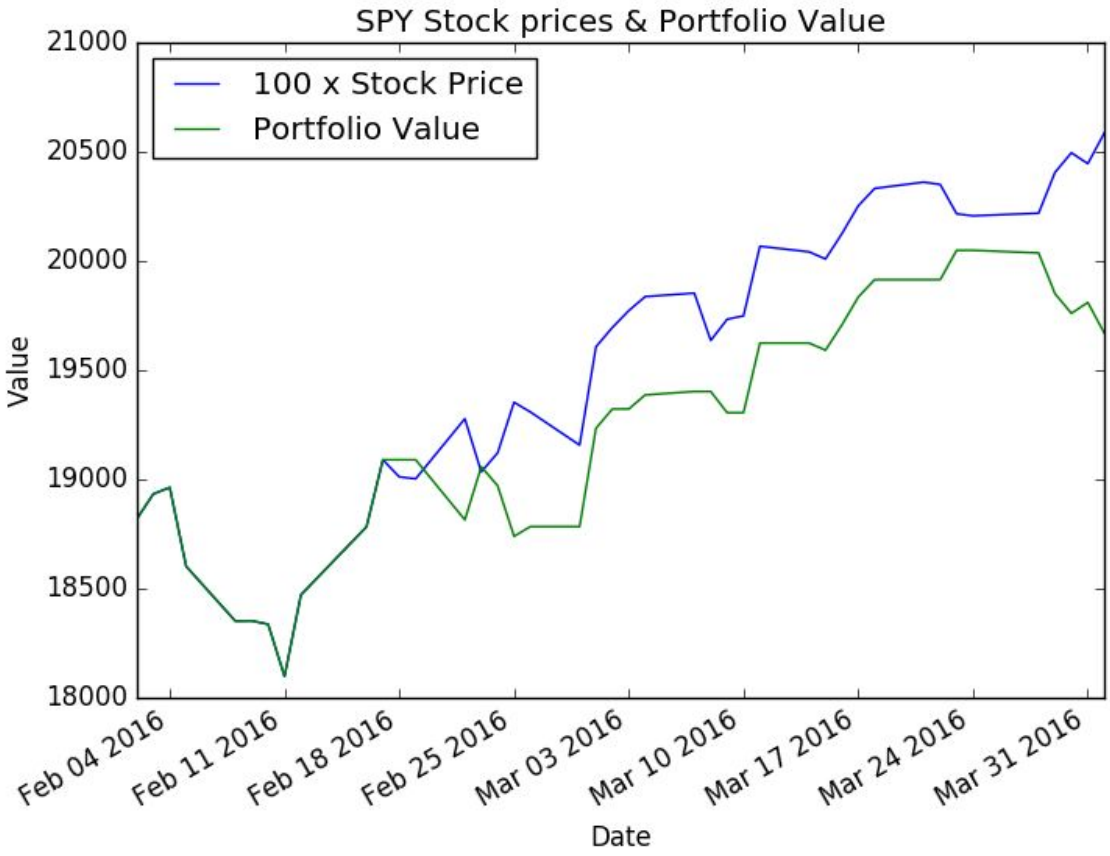
with a randomly initialized q-table, our initial cumulative return is 6.31%. As can be observed from the graph, training performance significantly improved with training, until converge at around 160.67%

Figure 2: comparing portfolio value with the market after training



As shown in the graph above, after training, the learner has a very good understanding of the market and significantly outperformed the market. This result is consistent through all five stocks.

Figure 3:result for applying learner to unseen data



The performance for the unseen data was not satisfactory.
Many factors could be further adjusted: definition of the state variable, more complicated trading scenario, diversification of the portfolio among multiple stocks etc.

5. Conclusion

Using daily portfolio value change as the reward, the q learner was effectively trained to trade optimally within the historical training data.
Considering the program was given no further instructions other than to maximize reward, i think the result is quite remarkable.
Even though it currently does not apply well to unseen data, it has been shown in hedge fund and researches that with adjustments and enhancement, this method will lead to optimized trade strategy which will lead to profit.

6. Python program

Program too big, Zip file enclosed.