CS7646 Project 8 Report

GTID: gyan31

System Description

The author implemented this machine learning trading system by using Q Learning. By construct states from discretized indicators of stock daily closed prices, and using daily portfolio changes as a reward, the system executes action suggestions returned from the trained Q Learner after inputting the states and reward. The following section will give detailed notes on how the system is designed and implemented.

1.1 Q Learning problem

Q Learning as a model-free machine learning method, designed to solve MDP problems. To use Q Learner to help trade on the stock market, we can process the stock market trading as an MDP problem. The next day's portfolio change only depends on the previous day's stock market and actions took.
In our case,

- we use discretized stock to adjust closed price indicators as a source of our **state**, and actions to take are Long, Short and Do Nothing while the maximal allowed absolute position number limited to 1000. While
- the **rewards** are price changes multiplied by the position held on the day.

By training the Q Learner on historical data, we can acquire suggested market position to take in the future tradings. Notice that we did not implement memory replay or Dyna in this experiment because the cost of inquiring the rewards and implement action is low since we are pulling states and rewards from historical data.

1.2 Discretization

The indicators we use in this study are price/SMA(simple moving average) ratio and TSI (True Strength Indicators) of SPY as explained in the report of Project 6. The indicators are continuous float numerical while our states are represented by discredited integral positive numbers, to convert the indicators to states, we need to discretize them first.

In this setup, the author used the Python Library Pandas' qcut module to discretize the indicators.
According to the Pandas document (https://pandas.pydata.org/pandas-docs/version/0.23.0/generated/pandas.qcut.html), the qcut module is a quantile-based discretization function. Discretize variable into equal-sized buckets based on rank or based on sample quantiles. For example, 1000 values for 10 quantiles would produce a Categorical object indicating quantile membership for each data point.
After the two indicators were discretized, we are assigning the indicators to a double-digit positive integral number with each digit represents an indicator. For our 2 indicator case, there would be 100 different states.
The generated categorical bins were saved by the system and reused when discretizing the indicators in predicting phase. Note that the data range of the predicting period could exceed the boundary of our bins, in this case, we would simply assign them to their corresponding closed ranks, e.g. category 11 would be reassigned to category 10, and category 0 would be reassigned to category 1.

1.3 System work-flow

This section describes how the system works in detail, and gives pseudo-codes to further explain how the different parts were implemented.
First, initialize the system by providing a market impact number.
Then train the system by invoking addEvidence method with parameters like stock symbol, start date, end date and starting value(optional).
After the training finished, we can use the system to infer suggested trading positions by involing the tesPolicy method supplied with stock symbol, time period and start value(optional).

1.3.1 addEvidence()

This method pulls the required stock price data and train the system. The psuedo codes are listed follows with comments:

```
def addEvidence(symbol,start_date,end_date,start_value):
    #0. Get required stock price data
    price=get_data(symbol,start_date,end_date)
    price_SPY=get_data(start_date,end_date)
    #1. Calculate indicators
    priceSMA=calculate_indicators(price)
    TSI_SPY=calculate_indicators(price_SPY)
    #2. Set up the Q Learner and discretize the indicators
    state_numbers,state_list=discretize(priceSMA,TSI_SPY)
    #2.1 There are 3 actions, 0 is Do Nothing, 1 is Long, 2 is Short
    QLearner.init(num_of_states,num_of_actions)
    #3 Start training
```

```
Loop=80
While  Policy_change_ratio > 0.05 and loop >1:
#3.1 The converge criteria would be either the policy basically stays the same or maximum number of loops reached.
    QLearner.state=day0.state
    QLearner.action=0
    holdings=0
    #3.2 We use day 0 as the initial day and start walk the price data from the second day day1.
    for i in range(1,end_date):
        #3.3 Compute next state and current reward
        state=day[i].state
        reward=calculate_reward(day[i],holdings)
        #3.4 Train the learner and implement the returned action suggestion
        QLearner.train(state,reward)
        action_suggested=QLearner.action
        if action==1 and holdings<1000:
            holdings+=1000
            price=impact(price)
        if action==2 and holdings >-1000:
            holdings-=1000
            price=impact(price)
    loop-=1
    policy_change=calculate_policy_change
```

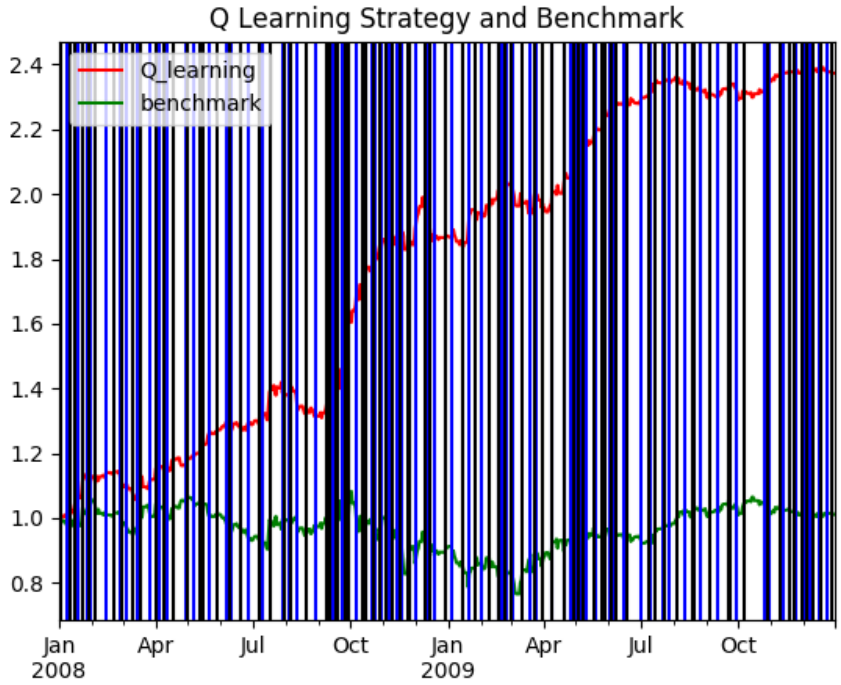
1.3.2 testPolicy()

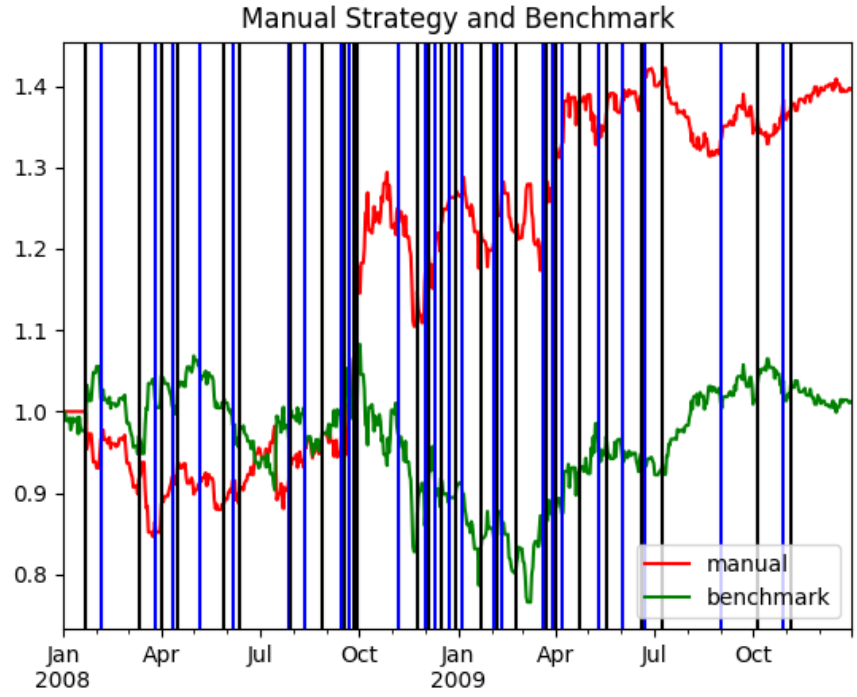
Psuedo codes as follows:

```
def testPolicy(symbol,start_date,end_date):
    holdings=0
    for day in range(start_date,end_date):
        state=calculate_sate()
        action=QLearner.suggest(state)
        if action==1 and holdings<1000:
            holdings+=1000
            policy.append(1000)
        if action==2 and holdings >-1000:
            holdings-=1000
            policy.append(-1000)
    return policy
```

2.1 Experiment 1 Performance Comparison with Manual Strategy

As required by the instructions, experiment 1 uses the same indicators as in project 6 in hope to compare the results of the two different ways of trading.
The experiment is trading JPM in-sample starting from 2008.1.1 to 2009.12.31 with the market impact of 0.005 with commission fees of 9.95 dollars per transaction same as Project 6.
The result images with benchmark depict the same way in Project 6 are posted below with a table of statistics.





Result Statistics

	Manual Strategy	Q Learning	Benchmark
Cumulative Return	0.395509	1.372528	0.012324
Mean of Daily Return	0.000523	0.001216	0.000116
Std. of Daily REturn	0.011574	0.008110	0.014155

As the results suggest, the Q Learner performs much better than the manual strategy. And the author believes that such relative result would be so every time within-sample tests. Because intuitively, compared with the manual strategy that provides at most dozens of rules for different cases, the Q Learner can provide suggestions for every possible case which is sure to outperform the manual strategy in every other in-sample performance.

2.2 Experiment 2 How Does the Market Impact Impact

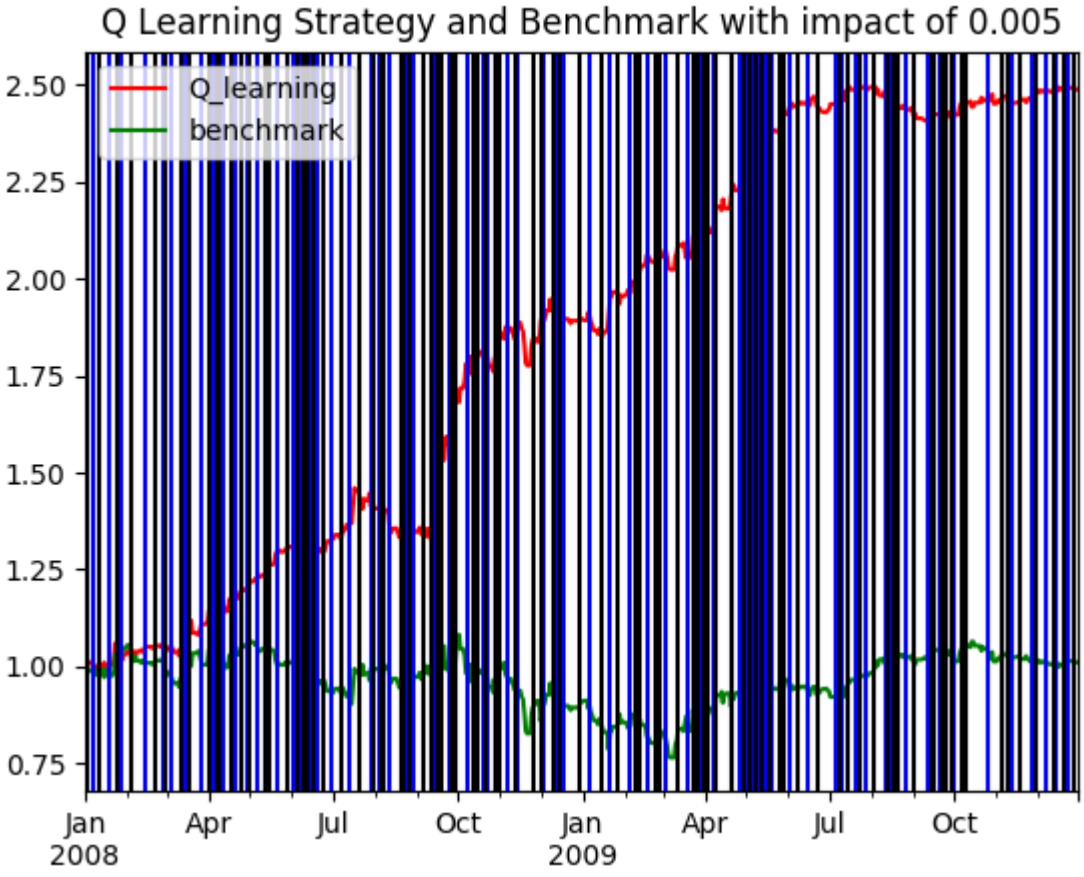
The hypothesis: The bigger the impact, the worse the System would perform.
In-sample period: JPM, 2008.1.1-2009.12.31
Impacts: 0.005, 0.03, 0.055, 0.08

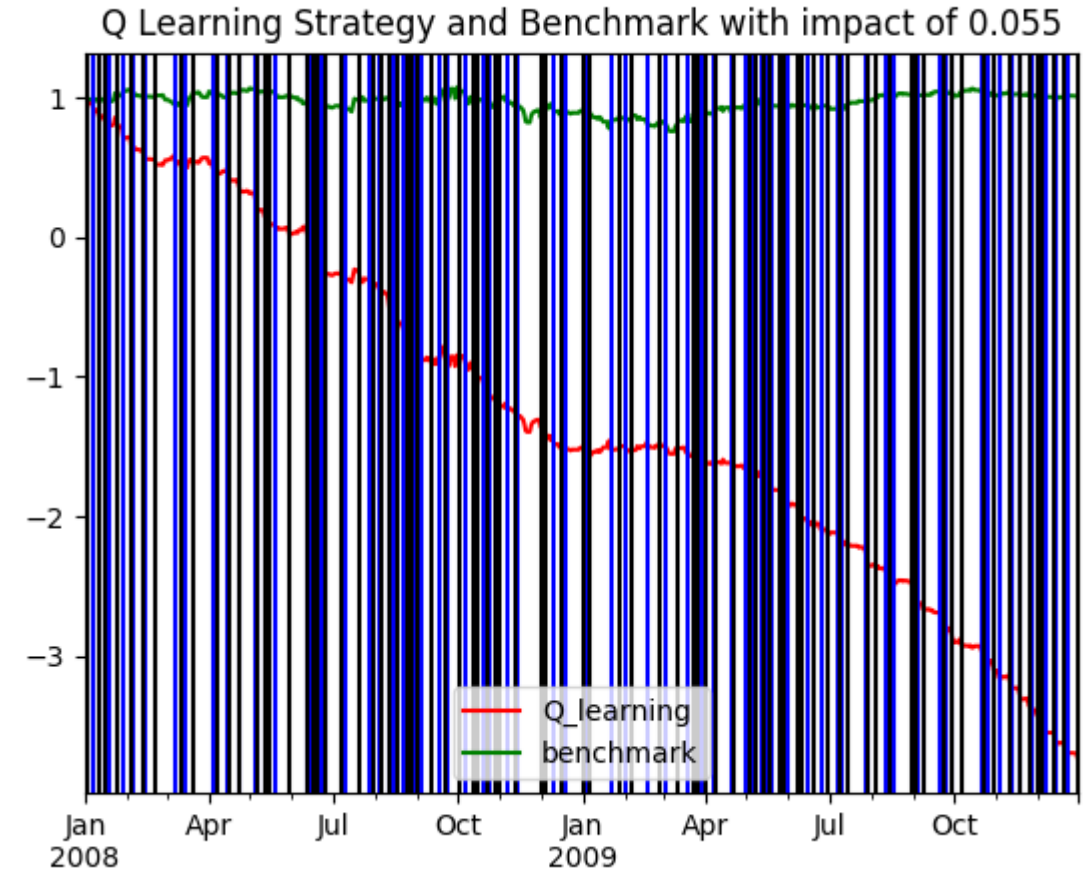
And the results have proven our claim.

Resulting statistics:

Impact	Cumulative Return	Mean of Daily Return	Std. of Daily Return
0.005	1.486592	0.001280	0.008013
0.03	-1.270912	-0.030997	0.512291
0.055	-4.743898	-0.012036	0.420598
0.08	-6.391272	0.003956	0.139435

Sample Graphs:


Q Learning Strategy and Benchmark with impact of 0.005
Q_learning
benchmark
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
Jan
2008
Apr
Jul
Oct
Jan
2009
Apr
Jul
Oct


Q Learning Strategy and Benchmark with impact of 0.055
Q_learning
benchmark
1
0
−1
−2
−3
Jan
2008
Apr
Jul
Oct
Jan
2009
Apr
Jul
Oct