Project 8: Strategy Learner

GTID Student Name: Zhiyong Zhang

GT User ID: zzhang726

GT ID: 903370141

Project Overview:

In this project, a reinforcement Learner-based approach of creating a Q-learning-based strategy was designed as a learning trading agent. The learner work in the following way:

The exactly the same indicators as in the manual strategy project was used. There are no changes to lookback windows or any other pertinent parameters.

In the training phase (e.g., addEvidence()), the learner is provided with a stock symbol and a time period. It uses this data to learn a strategy.

In the testing phase (e.g., testPolicy()), the learner is provided a symbol and a date range. All learning is turned off during this phase. If the date range is the same as used for the training, it is an in-sample test. Otherwise it is an out-of-sample test. The learner return a trades dataframe like it did in the last project.

In this report, the symbol traded is JPM. The in sample/development period is January 1, 2008 to December 31 2009. And the out of sample/testing period is January 1, 2010 to December 31 2011. Starting cash is $100,000. Allowable positions are: 1000 shares long, 1000 shares short, 0 shares. Commission is $0.00 while Impact may vary.

The performance of a portfolio starting with $100,000 cash, investing in 1000 shares of the symbol in use and holding that position are used as Benchmark.

A learner that can learn a trading policy using the learner is developed. Q-Learner from the earlier project is used directly to Implement Strategy Learner. The Strategy Learner implements the following API:

```
import StrategyLearner as sl
learner = sl.StrategyLearner(verbose = False, impact = 0.000) # constructor
learner.addEvidence(symbol = "AAPL", sd=dt.datetime(2008,1,1), ed=dt.datetime(2009,12,31),
    sv=100000) # training phase
df_trades = learner.testPolicy(symbol = "AAPL", sd=dt.datetime(2010,1,1),
    ed=dt.datetime(2011,12,31), sv = 100000) # testing phase
```

The input parameters are:

- verbose: if False do not generate any output
- impact: The market impact of each transaction.
- symbol: the stock symbol to train on
- sd: A datetime object that represents the start date
- ed: A datetime object that represents the end date
- sv: Start value of the portfolio

The output result is df_trades: A data frame whose values represent trades for each day. Values of +1000 indicates a BUY of 1000 shares, -1000 indicates a SELL of 1000 shares, and 0.0 indicates do nothing. The net holdings are constrained to -1000, 0, and 1000.

Qlearner Framing

For this project, Bollinger Band indictor (bbi) and three stocking holding (-1 = short, 0 = no holding,1 = long) are used to set state s. For the learning period, bbi is sorted, then divided to N_bins (10 in this project), the bin number n_bbi (0 to N_bins-1) defines which bin the bbi is in. Then the Qlearner state **s** is defined as

s(bbi, holding)= n_bbi * 3 + (holding + 1)

Qlearner action (-1 = sell, 0 = nothing,1 = buy) is defined as

action = learner.query(state) – 1

reward **r** is calculated every day based on stock price movement, action taken and impact

if nothing
r = (prices[n+1]-prices[n])*holding
if sell
r = (prices[n+1]-prices[n])*(-1) - impact*prices[n]
if buy
r = (prices[n+1]-prices[n])*(1) - impact*prices[n]

Qlearner Initialization

The constructor QLearner() reserve space for keeping track of Q[s, a] for the number of states and actions. Q is initialized with all zeros. Details on the input arguments to the constructor:

```
learner = ql.QLearner(num_states = 100, \
        num_states=100, \
        num_actions = 4, \
        alpha = 0.2, \
        gamma = 0.9, \
        rar = 0.5, \
        radr = 0.99, \
        dyna = 0, \
        verbose = False)
```

num_states integer, the number of states to consider
num_actions integer, the number of actions available.
alpha float, the learning rate used in the update rule.
gamma float, the discount rate used in the update rule.
rar float, random action rate: the probability of selecting a random action at each step.
radr float, random action decay rate, after each update, rar = rar * radr.
dyna integer, conduct this number of dyna updates for each regular update.
verbose boolean, whether allowed to print debugging statements

Qlearner Learning

The in sample data are used to train the learner. The processes is repeated multiple times until the total reward converges.

query(s', r) is the core method of the Q-Learner. It should keep track of the last state s and the last action a, then use the new information s' and r to update the Q table using following formula:

$$Q[s, a] = (1 - alpha) * Q[s,a] + alpha * (r + gamma * argmax_{a'} Q[s',a'])$$

The learning instance, or experience tuple is <s, a, s', r>. query() should return an integer, which is the next action to take. Note that it chooses a random action with probability rar, and that it updates rar according to the decay rate radr at each step. Details on the arguments:

s', integer, the new state.
r float, a real valued immediate reward.

querysetstate(s) is a special version of the query method that sets the state to s, and returns an integer action according to the same rules as query(), but it does not execute an update to the Q-table. It also does not update rar. There are two main uses for this method: 1) To set the initial state, and 2) when using a learned policy, but not updating it.

Experiment 1:

Exactly the same Bollinger Band indictors (bbi) and the same impact (.005) are used are used in manual_strategy (trade JPM) and Strategy Learner. The results are showed on Figures 1-2. and in Table 1.

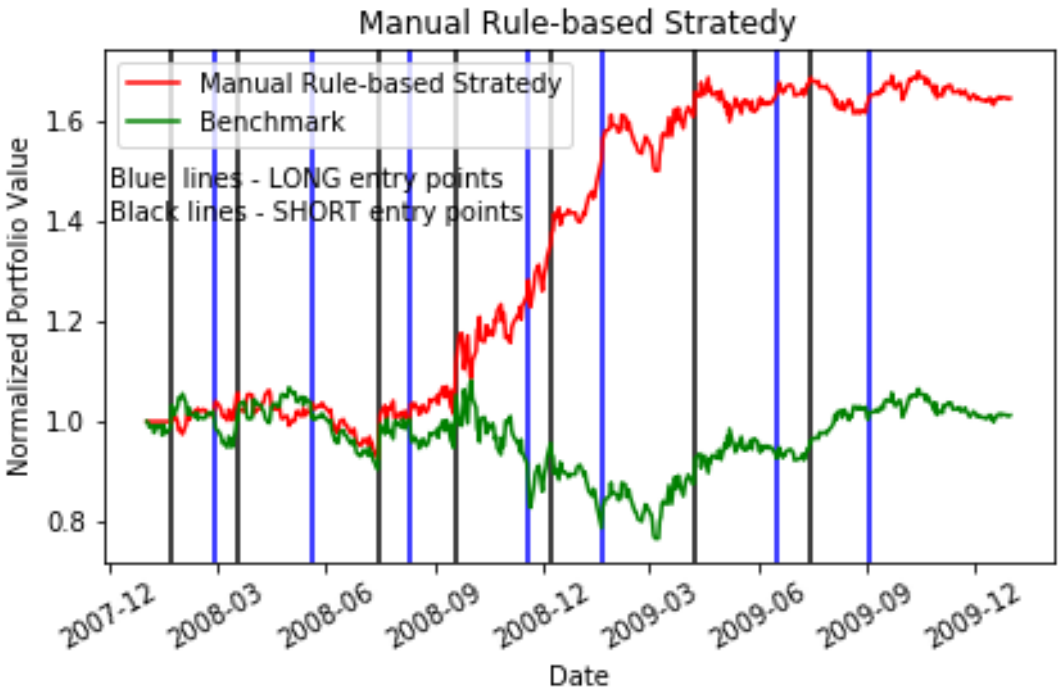

Figure 1. Performance of benchmark and manual rule-based portfolio.

Figure 2. Performance of benchmark and Strategy Learner.

	Cumulative return	Standard Deviation of daily returns	Mean of daily returns
Benchmark	0.012	0.0170	0.00017
Manual rule-based	0.648	0.0127	0.00107
Strategy Learner	**0.430**	0.013	0.0008

Table 2. The performance summary of benchmark and manual rule-based portfolio.

Both of Manual rule-based and Strategy Learner perform better than Benchmark. However, Manual rule-based out perform Strategy learner when using impact of 0.005.

We would expect both Manual rule-based and Strategy Learner to perform better than Benchmark But there is no guarantee which one perform better between Manual rule-based and Strategy Learner.

Experiment 2

High impact would reduced the performance of Strategy Learner based trading. The reason is that high impact increase the cost of selling and buying and decrease the rewards.

Figure 3 and Table 2 show the performance of trading strategy with different impact (0-0.005). It shows that generally higher impact have lower cumulative return.

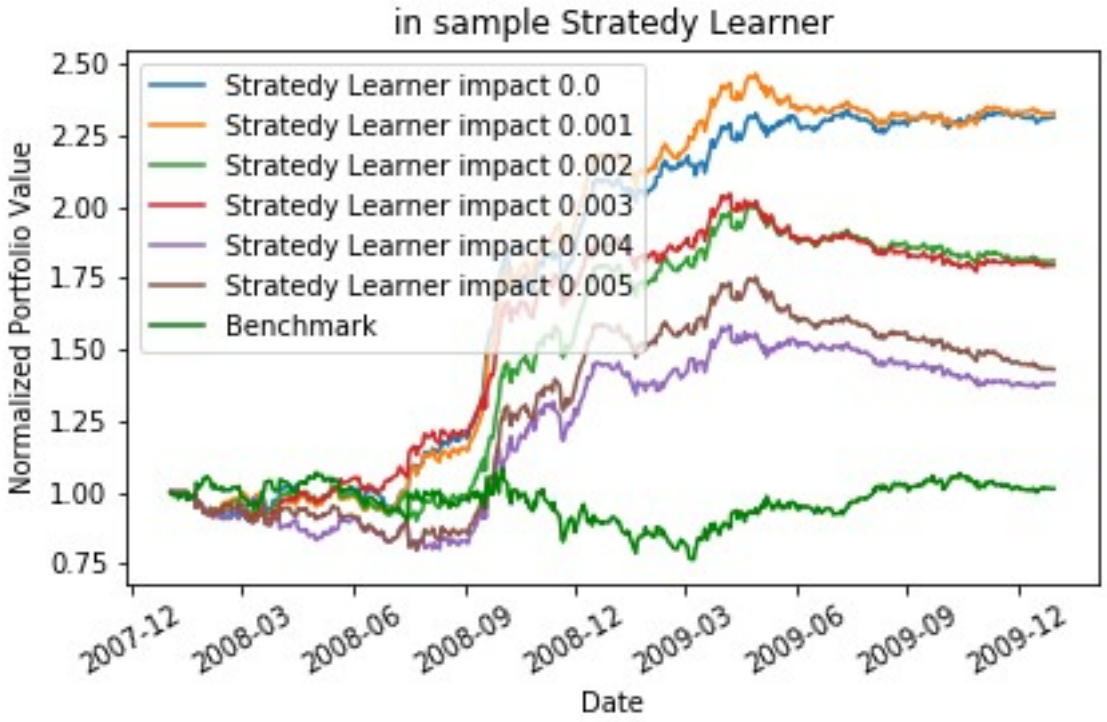

Figure 3. Performance of benchmark and Strategy Learner trading.

Case	impact	Cumulative return	Standard Deviation of daily returns	Mean of daily returns
In sample Benchmark	0.005	**0.012**	0.017	0.0002
In sample Manual rule-based	0.005	**0.648**	0.013	0.0011
In sample Strategy Learner	0.005	**0.430**	0.013	0.0008
	0.004	**0.378**	0.014	0.0007
	0.003	**0.793**	0.011	0.0012
	0.002	**0.809**	0.012	0.0012
	0.001	**1.328**	0.010	0.0017
	0	**1.313**	0.011	0.0017

Table 3. The performance summary of benchmark and Strategy Learner trading for in sample periods.