Jiawei Zhang jzhang950
Part 1: StrategyLearner
A. Description:
I selected the Q-learner as my Reinforcement Learner-based approach. To frame the structure of trading problem, I divided the StrategyLearner mainly in training and querying two sessions as followings in general:
-AddEvidence (training part):

1. I selected the ratio of SMA over price, Momentum and Bollinger Bands as my indicators.
2. In order to create a state, I chose integer 10 to discretize the features of indicators such that the state can be constructed by:
   indicator['sma_over'] * 100 + indicator['momentum'] * 10 + indicator['bb']
3. It is necessary for me to set the state type as in before using the state:
   state = state.astype(int)
   Since the state is a 3-digit number, so the num_states for me to initialize the Q Learner is 1000 for maximum and I preferred the num_actions = 3 because as requirement asked, there can be three types of actions during trading strategy learner: SHORT, LONG and CASH.
4. To train a strategy learner by a Qlearner, I create a dataframe which has four columns 'Hold', 'Price', 'Cash', 'Portfolio' and initialize the first row of 'Cash', 'Portfolio' to be the start value of the cash.
5. Then I begin to train the data while the policy is converged. I for looped 13 times here. The general steps can be described as selecting the current state, computing the reward for the last action, querying the learner with the current state and reward to get an action, implementing the action the learner returned, and updating portfolio value
   To implement the instructions above:

```
for i in range (0, 13):
        s = state.ix[0, syms]
        a = self.learner.querysetstate(s)
        holding = 0
        for j in range (1, state.size):
            number = np.absolute(Q_frame.ix[j, 'Hold'])
            rate = Q_frame.ix[j, 'Price']
            if holding == 0 and a == 1:
                holding = 1
                Q_frame.ix[j, 'Hold'] = -1000
                Q_frame.ix[j, 'Cash'] = Q_frame.ix[j-1, 'Cash'] - Q_frame.ix[j, 'Price'] * Q_frame.ix[j, 'Hold']
            elif holding == 0 and a == 2:
                holding = 2
                Q_frame.ix[j, 'Hold'] = 1000
                Q_frame.ix[j, 'Cash'] = Q_frame.ix[j-1, 'Cash'] - Q_frame.ix[j, 'Price'] * Q_frame.ix[j, 'Hold']
            elif holding == 1 and a == 2:
                holding = 2
                Q_frame.ix[j, 'Hold'] = 1000
                Q_frame.ix[j, 'Cash'] = Q_frame.ix[j-1, 'Cash'] - Q_frame.ix[j, 'Price'] * Q_frame.ix[j, 'Hold'] * 2
            elif holding == 2 and a == 1:
                holding = 1
                Q_frame.ix[j, 'Hold'] = -1000
                Q_frame.ix[j, 'Cash'] = Q_frame.ix[j-1, 'Cash'] - Q_frame.ix[j, 'Price'] * Q_frame.ix[j, 'Hold'] * 2
            else:
                Q_frame.ix[j, 'Hold'] = Q_frame.ix[j-1, 'Hold']
                Q_frame.ix[j, 'Cash'] = Q_frame.ix[j-1, 'Cash']
```

```
Q_frame.ix[j, 'Portfolio'] = Q_frame.ix[j, 'Cash'] + Q_frame.ix[j, 'Price'] * Q_frame.ix[j, 'Hold']
reward = Q_frame.ix[j, 'Portfolio'] / Q_frame.ix[j-1, 'Portfolio'] - 1
s = state.ix[j, syms]
a = self.learner.query(s, reward)
```

Note:
Where the parameter holding here is to indicate the holding constraint. holding = 0 means that there is no shares current so that we can at most BUY or SELL 1000 shares. holding = 1 means that holdings is -1000 shares and we can BUY 2000 shares for holding = 1000. On the contrary, holding = 1 means that there is 1000 shares and we can at most to SELL 2000 shares to get the most profits.
Also, action = 1 means SELL. Action = 2 means BUY and action = 0 means HOLD state.
For us to execute each action in the market, we will update the 'Cash' column and finally use the 'Cash' column to calculate the 'Portfolio' column. Then we can use the result to get the reward and get a new action by pushing the new updated state and reward to call the query function in QLearner.

-TestPolicy (querying part):

1. The first step of testing is still using the indicators to get state as addEvidence.
2. Then we create a dataframe to generate the action in market.
3. For each day in the testing data: computing the current state by passing the parameter into querysetstate function in QLearner. Then querying an action. Implementing the action the learner returned (LONG, CASH, SHORT).

```
holding = 0
for i in range(1, state.size):
    s = state.ix[i-1, [symbol]]
    a = self.learner.querysetstate(s)
if holding == 0 and a == 1:
    holding = 1
    trades.ix[i, 'value'] = -1000
elif holding == 0 and a == 2:
    holding = 2
    trades.ix[i, 'value'] = 1000
elif holding == 2 and a == 1:
    holding = 1
    trades.ix[i, 'value'] = -2000
elif holding == 1 and a == 2:
    holding = 2
    trades.ix[i, 'value'] = 2000
else:
    holding = holding
```

Note:
action = 1:SELL. action = 0: HOLD. action = 0: BUY.
holding is also the meaning of holding shares as specified in the note in the addEvidence function to determine how much shares that the investors can trade in this trading.

B. indicators

As I mentioned in the discription, I selected SMA, Momentum and Bollinger Bands as my indicators. Then I use the integer 10 to dicretizatize to generate a state.

The function is X1*10^2+X2*10+X3. At the beginning, I set the X1 is SMA ratio, X2 is Bollinger Bands and X3 is Momentum. However, when I tried to test my strategy learner, I found the strategy cumulative return is negative, which cannot beat the benchmark. So, I changed my strategy with X1 is SMA ratio, X2 is Momentum and X3 is Bollinger Bands. It works. The reason that I think make the differences between two results is value of BB sometimes cannot be a determination value since adding 10*BB does not make any big difference for all the value. So, the action will not change for a big difference. In other words, the level of discretization is not much as expected.

Part 2: Experiment1

1. Input

symbol = “JPM”

sd=dt.datetime(2008,1,1)

ed=dt.datetime(2009,12,31)

sv = 100000

commission = 9.95, impact = 0.005 for marketsim

2. Procedure

There are three main parts:

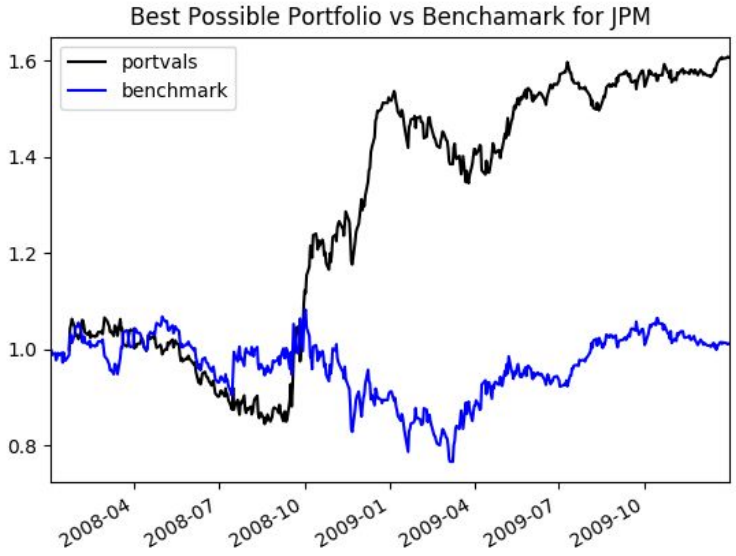
a) Manual Strategy: Call ManualStrategy.py to calcculate the cumulative return in marketsim.py to plot.

b) Benchmark: BUY 1000 shares in the first day and SELL 1000 shares at the last day.

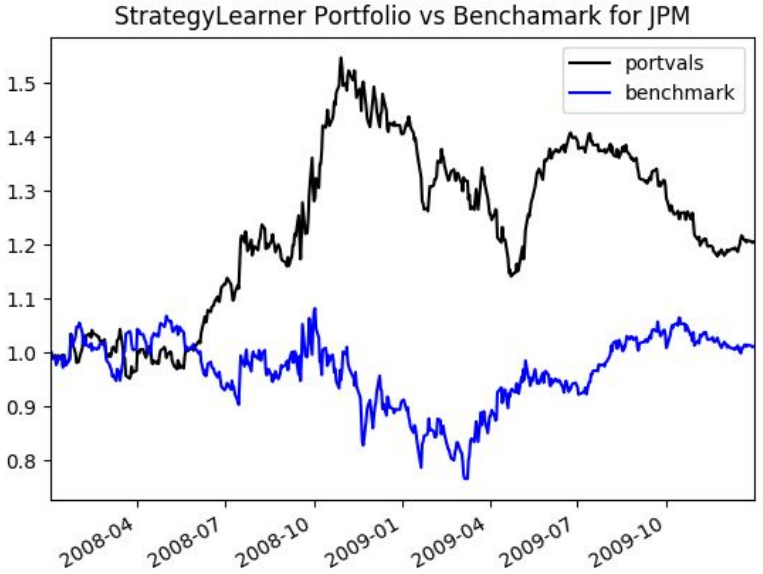
c) Strategy Learner: In this part, I call the StrategyLearner.py to addEvidence and then testpolicy function to to test an in sample data. Since the output of data frame is not in the format of the marketsim input format. I also transfer the formation as followings to compute the cumulative return and plot:

```
Qframe = pd.DataFrame(index = frame.index, columns = ['Symbol', 'Order', 'Shares']);
Qframe.ix[0:, 'Symbol'] = "JPM"
for i in range (0, len(frame)):
    if frame.ix[i, 'value'] < 0:
        Qframe.ix[i, 'Order'] = 'SELL'
        Qframe.ix[i, 'Shares'] = np.absolute(frame.ix[i, 'value'])
    else:
        Qframe.ix[i, 'Order'] = 'BUY'
        Qframe.ix[i, 'Shares'] = np.absolute(frame.ix[i, 'value'])
Qframe = Qframe[Qframe.Shares != 0]
```

3. Outcome:



Manual Strategy:
Cumulative Return: 0.6069
Average Daily Return: 0.00106666470382
Standard Deviation Daily Return: 0.014218021926
Benchmark:
Cumulative Return: 0.0123
Average Daily Return: 0.000168086978191
Standard Deviation Daily Return: 0.0170043662712



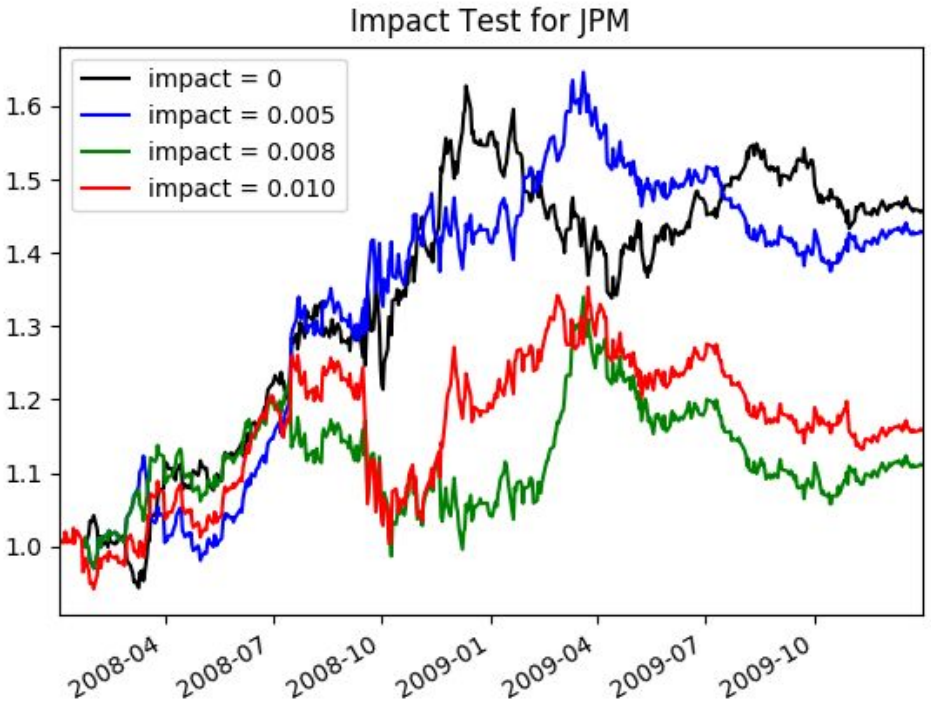
Strategy Learner:
Cumulative Return: 0.2062
Average Daily Return: 0.000455777065273
Standard Deviation Daily Return: 0.0129156036261

Note: The result (cumulative return) of the StrategyLearner is not as good as the ManualStrategy since the ManualStrategy we provided a best possible strategy. Since action is random in our Qleanrer. There may be not as good as we expected. However, the cumulative return of StrategyLearner should always beat the that of benchmark.

4. Expectation:
As I mentioned, the Qlearner accepts the random number for action taken. The dataframe that the StrategyLearner returned cannot always be the same. However, the result is supposed to be relative with the in sample data since after running the many times, I found the graph is almost in the same style. One thing that can be sure is that the result of StrategyLearner always beats the benchmark.

Part 3: Experiment2
Hypothesis: The larger impact we select, the smaller cumulative return of in sample data we will get because the impact will result the cash value in every time of trading: Cash = Cash - impact * (number of shares * price). Since the portfolio is resulting from the Cash and the reward is calculated from portfolio. So, the reward may be changed due to the larger impact value and turns to be a negative. The action, of course, in the QLearner will changed due to we increase the impact value.

To test my hypothesis, I generated two sets of comparisons and the results are:
1. impact = 0, 0.005, 0.008, 0.010



Cumulative Return for impact = 0: 0.4573

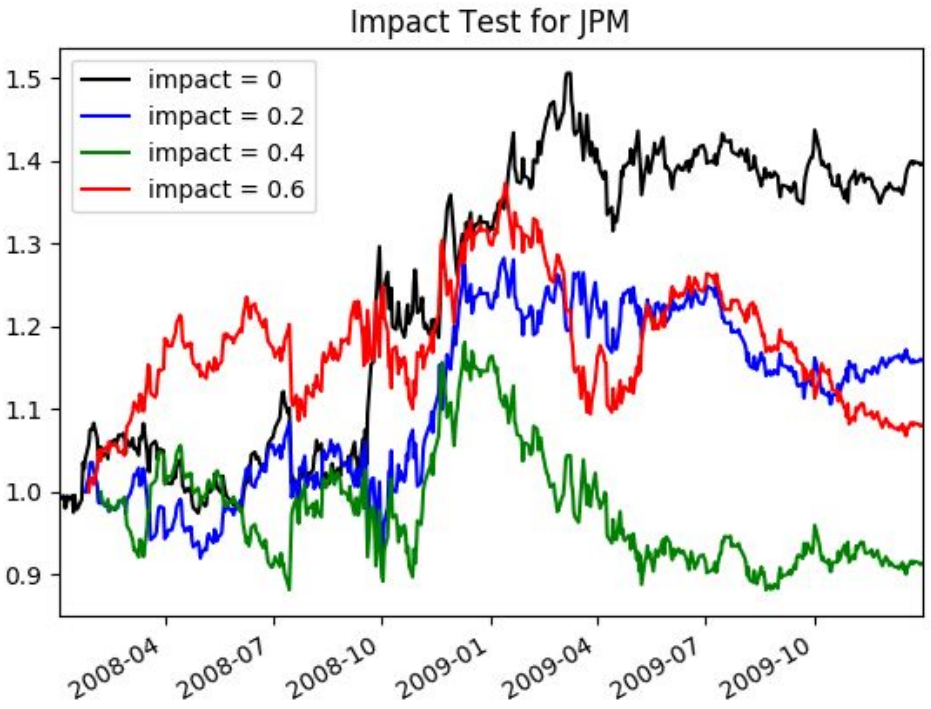
Average Daily Return: 0.00084443899743

Cumulative Return for impact = 0.005: 0.4277
Average Daily Return: 0.000798995650285

Cumulative Return for impact = 0.008: 0.1103
Average Daily Return: 0.000317793951558

Cumulative Return for impact = 0.010: 0.1582
Average Daily Return: 0.000389631821081

3. impact = 0, 0.2, 0.4, 0.6



Cumulative Return for impact = 0: 0.3964
Average Daily Return: 0.000752906697639

Cumulative Return for impact = 0.2: 0.1585
Average Daily Return: 0.000410962438725

Cumulative Return for impact = 0.4: -0.0861
Average Daily Return: -5.42442167794e-05

Cumulative Return for impact = 0.6: 0.0809
Average Daily Return: 0.000251832641352

In general, we noticed that the greater impact we choose, the smaller cumulative return as well as the average daily return we will get. The reward value is sometimes negative as may change the action value several times because the cost increases.