CS7646 MC3-Project-4 Report

Kaidi Zhang (kzhang346)

Description of Strategy Learner

My trading Strategy Learner is a Q-learner based strategy. Here are the meanings of the four components in my Q-learner expression tuple Q<s, a, s', r>:

- **s or state:** technical indicators of the current stock price. This is calculated in getStates function.

  The three indicators I used are SMA ratio, Bollinger Band percentage and relative strength index (RSI), same as Professor David Byrd used in "vectorize me" lecture.

  SMA ratio is current price divided by simple moving average. When SMA ratio is above 1, the current stock price is higher than the moving average.

  Bollinger Bands percentage = (current price – bottom Bollinger Band) / (top Bollinger Band – bottom Bollinger Band). When the percentage is higher than 1, the current price is higher than top Bollinger Band, i.e., overbought. When the percentage is lower than 0, the current price is lower than bottom Bollinger Band, i.e., oversold.

  RSI is more complexed than the other two indicators. RSI indicates the speed and change of price movements. RSI = 100 – [100 / (1 + (Average of Upward Price Change / Average of Downward Price Change)). The stock is considered overbought when RSI is above 70 and oversold when RSI is below 30 (https://www.fidelity.com/learning-center/trading-investing/technical-analysis/technical-indicator-guide/RSI).

  All the three indicators need to have a lookback time period. I use the same lookback time window (21 days) when computing all the three indicators. To obtain the states for the first 21 days, I use util get_data function to get the stock prices from 2 * 21 days before the start date to the end date. Then after I calculate all the three indicators, I discretize their values into 10 bins using pandas.qcut function. pandas.qcut is quantile-

based discretization function. In other words, the values are discretized into equal-sized bins, compared with value-based pandas.cut. The next step is to combine the three indicators into one state: states = 100 * bbp + 10 * sma + rsi.

The final step in getStates is to calculate benchmark, which is simply the return of buying 200 shares on the first trading day and selling on the last trading day, minus the transaction cost. After removing data before the start date, states, daily_rets (price difference between current day and previous day), prices and benchmark are returned from getStates.

- **a or action:** holding of the stock. There are three actions: 0 shares (action 0), 200 shares LONG (action 1), 200 shares SHORT (action 2). Function actionToHolding is converting actions in 0, 1, 2 into stock holding.

- **s' or new state**: stock price indicators on the next day.

- **r or reward:** daily reward based on current stock holding and price change.
  For example, on day n-1, the action is 0, zero shares. And then on day n, the action returned by learner.query is 2 (200 shares SHORT). Reward is initialized as -1 * transaction cost. Transaction cost = commission cost + (holding on day n – holding on day n-1 absolute value) * stock price * market impact penalty. If the holding is the same on day n-1 and day n, reward is initialized as 0. Then on day n+1 (next iteration), we add holding (-200) * stock price change (price on day n+1 – price on day n) to the reward. This is the immediate reward on day n.

There are two main functions in Strategy Learner: addEvidence and testPolicy.

- **Function addEvidence** first calls getStates, and then creates a Q-learner and trains the Q-learner. The minimum number and maximum number of epochs that Q-learner can go through are 20 and 200, respectively. After 20 epochs, if the Q-learner has

converged, the training process will stop. The convergence is checked by comparing portfolio value on the last trading day between previous epoch and current epoch. Inside each epoch, I first call learner.querysetstate (state on the first trading day) to set the initial set and then iterate through all the rest days. In each iteration, I first update the reward value with holding * daily returns, get a new holding position by calling learner.query (new state, reward), and initialize a new reward with transaction cost. Through the iterations, I keep track of the portfolio value by adding up daily rewards.

- **Function testPolicy** is to test the trained Q-learner. testPolicy first calls getStates to get states, daily_rets, prices and benchmark, and then iterate through every trading day from start date to end date. This is similar as addEvidence, except that learner.querysetstate is used instead of learner.query so that the Q-table will not get updated. During the iterations, holding on every trading day is stored in a dataframe. The last step in testPolicy is to calculate and return trades: trade = holding on the next day – holding on the current day.

  I make some plots using normalized values of prices, daily returns, holdings, portfolio value, etc. **Fig. 1 and Fig. 2** are two plots from in-sample testing of ML4T-220 (January 1, 2008 to December 31 2009) after training. Fig. 1 shows stock prices, holding positions, and portfolio value. Stock holding position is changing together with the fluctuating stock price. When stock price is increasing, holding is positive (200 shares LONG). When stock price is decreasing, holding is negative (200 shares SHORT). Therefore, the portfolio value is in an increasing trend during the whole-time period. Fig. 2 is a scatter plot between stock holding position and daily returns. Holdings are positive (LONG) when daily returns are above zero. Holdings are negative (SHORT) when daily returns are below zero.

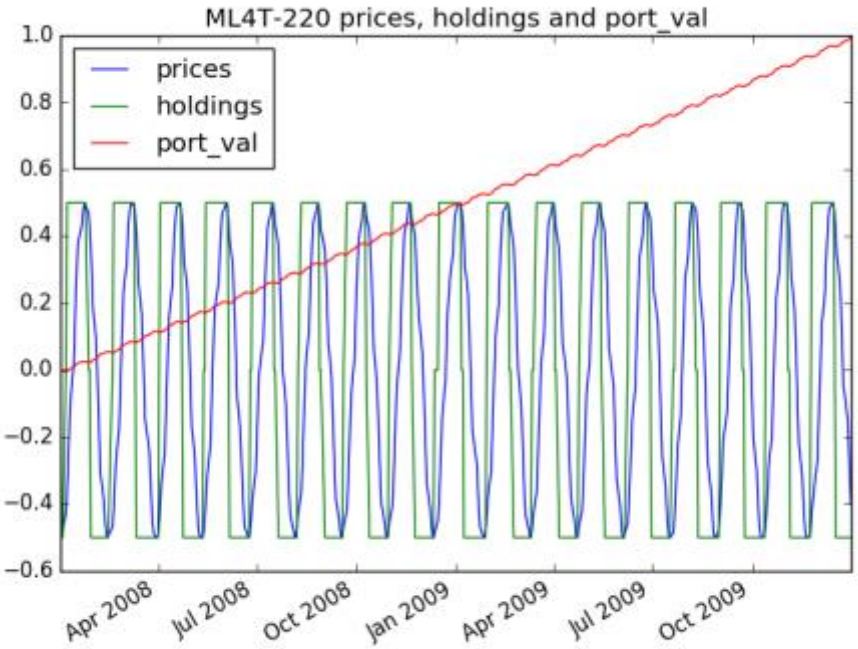

Fig. 1. ML4T-220 prices, holdings and portfolio values

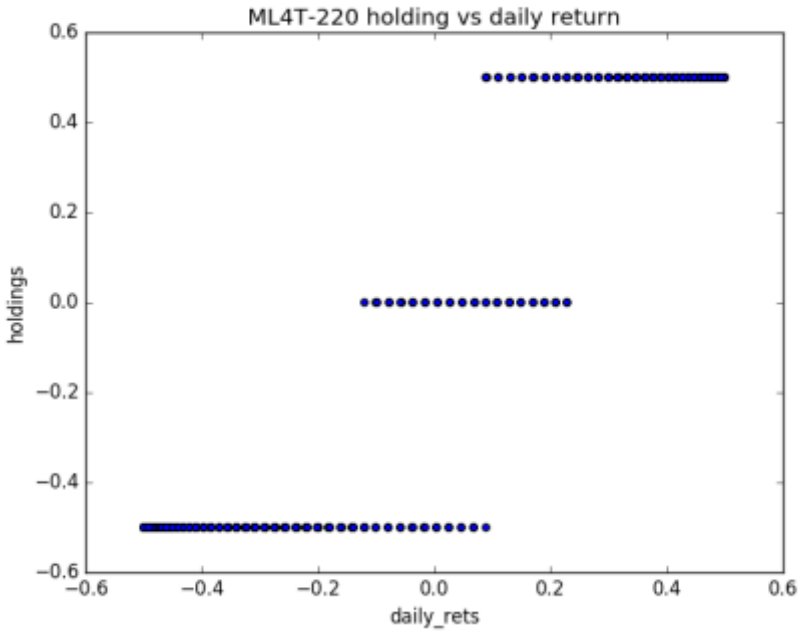

Fig. 2. ML4T-200 holdings vs daily returns

Experiment 1: choose lookback time window

The first experiment I perform is to check the impact of choosing different time windows on the Q-learner trading strategy. The computing of the three technical indicators all requires a lookback time period. Therefore, my question is what is the optimal lookback time window. I choose lookback periods from 15 days to 64 days and make plots of portfolio values on the last trading day (divided by start value) vs lookback time window.

Fig. 3 and Fig. 4 show the results from ML4T-220 and AAPL. Interestingly, the Q-learner trading strategy for ML4T-220 is doing a bad job (even losing money) when lookback time window is 42, 43 or 44 days. The reason is probably because the stock price is fluctuating every 42-43 days and the rolling mean and rolling standard deviation do not change at all. On the other hand, there doesn't seem to be an obvious relationship between AAPL final portfolio values and lookback time windows.

Based on the two figures, I choose 21 days as my lookback time window.

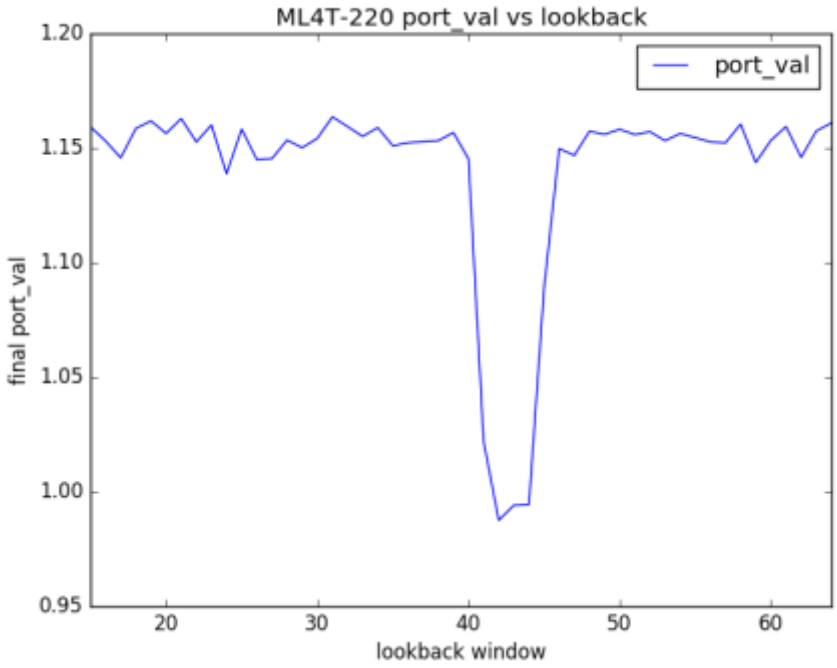

Fig. 3. ML4T-220 portfolio value vs lookback time window

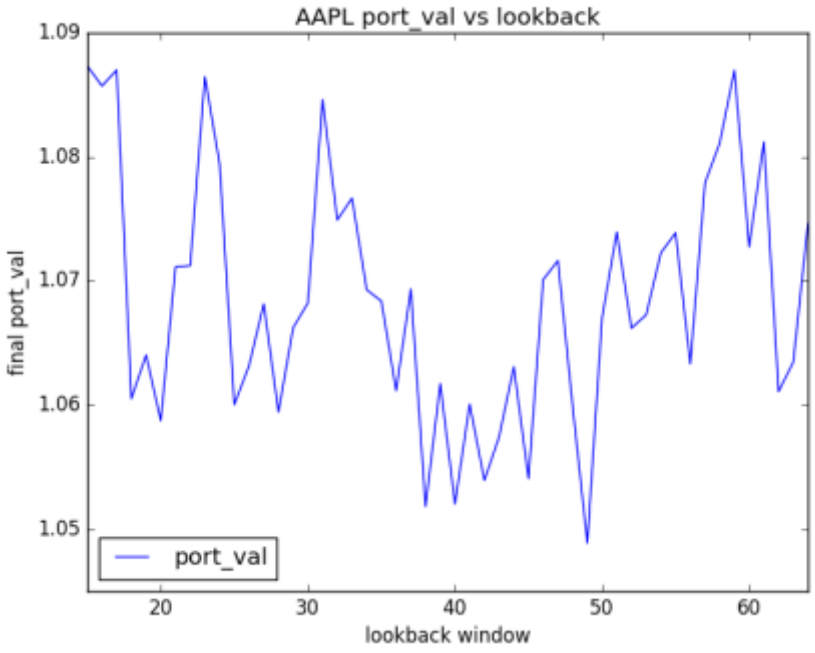

Fig. 4. AAPL portfolio value vs lookback time window

Experiment 2: choose random action rate

The second experiment I perform is to look at the effects of random action rate (rar) on the Q-learner. rar determines the probability of selecting a random action. This probability is high at the beginning of the training process and becomes lower and lower by multiplying the random action decay rate (radr = 0.999) at each step. This randomness forces the learner to choose random actions and explore different actions in different states. I choose rar from 0.01 to 0.99 and test it on ML4T-220 and AAPL.

Fig. 5 and Fig. 6 show the experimental results. There is a positive correlation between rar and the final portfolio value on both ML4T-220 and AAPL, shown by the two red fit lines. What is not included in the plots is that portfolio value will stay at the start value if rar is set at 0. Therefore, based on this experiment, I choose a high random action rate (0.98) for my Q-learner in Strategy Learner.

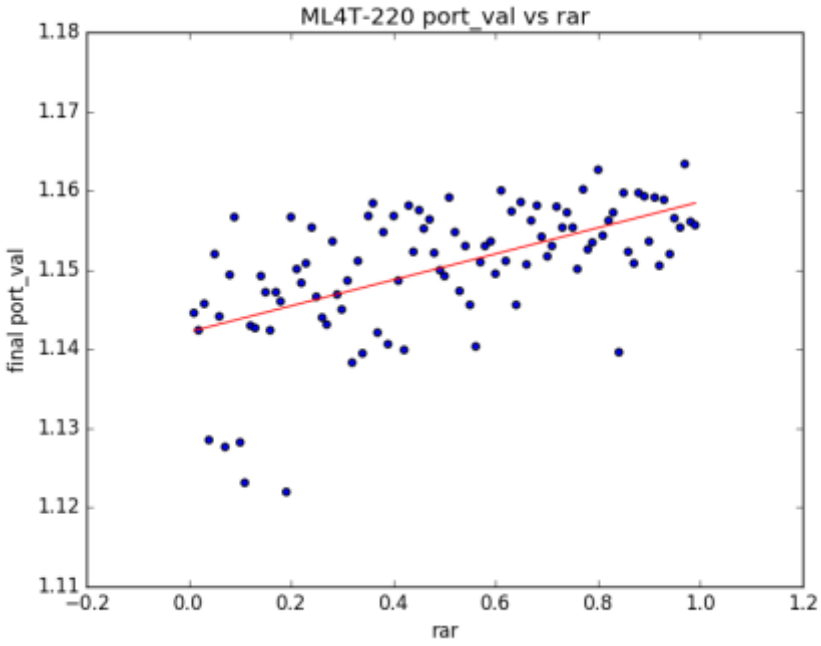

Fig. 5. ML4T-220 portfolio value vs rar

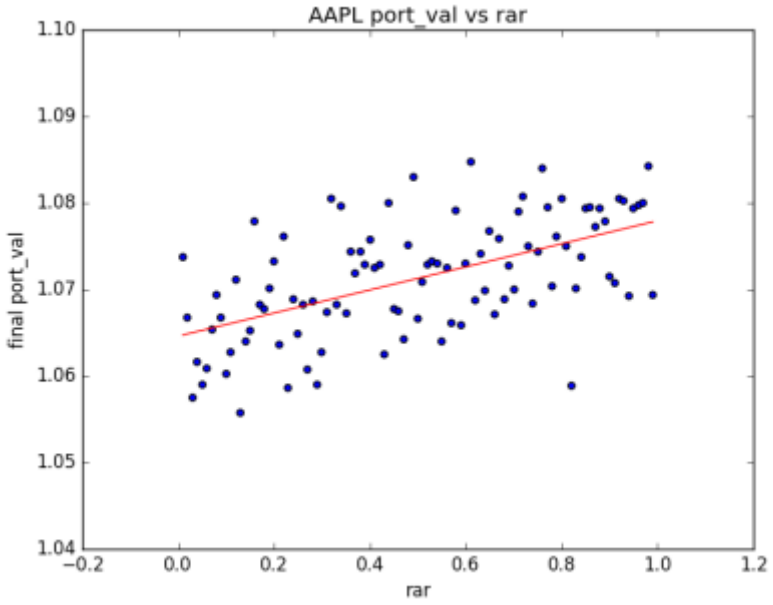

Fig. 6. AAPL portfolio value vs rar