PROJECT 8 STRATEGY LEARNER

Spring 2019

Jarod Kennedy

Jkennedy76

903369277

INDICATORS

The indicators utilized in the Strategy Learner are the same as those used in the Manual Strategy. Explanations of these indicators and their usefulness make up the bulk of the Manual Strategy report and therefore will not be included in this report.

As a reminder the indicators used in BOTH Manual Strategy and Strategy Learner are: on balance volume, price per simple moving average of price over a twenty-day rolling window, and Bollinger Bands, also calculated over a twenty-day rolling window.

FRAMING THE PROBLEM

CHOOSING A LEARNER

Of the choices between regression, reinforcement and classification-based learners, Q-learning was settled upon primarily in anticipation of runtime constraints. The time taken to run the Q-learner is highly predictable and can be reduced easily by simply reducing the number of epochs and decreasing the size of the Q-table. Cutting off the learner before it has fully converged is suboptimal, but, experience has shown that the Q-learner can provide reasonably effective results long before convergence.

UNREALISTIC AFFORDANCES AND ASSUMPTIONS

This project was not taken under real time conditions and therefore a few steps were taken to reduce runtime that would otherwise be suboptimal or completely impossible. Dyna was not implemented to hallucinate iterations of the Q-learner. With access to all the data for both the in and out of sample testing periods immediately, and no need to spend real money on trades,

implementing Dyna would have only slowed the program and provided little in terms of improved performance for this particular problem.

Given access to all the data ahead of time, indicators were calculated and discretized for the entire time span in question, before any data was fed to the Q-learner. Naturally this would not be possible on a system that is trading in real time.

Trades are made on closing prices; this is a limitation due to the information available. I do not have data on the stock prices from some minutes before close and therefore cannot act on that data. Naturally one cannot trade on the close price because the market is closed and deciding whether or not to make a trade takes time.

DISCRETIZATION, STATE BUILDING

Choosing exactly how to discretize the indicators such that the Q-learner might see the state as an integer was the crux of this trading problem. For the sake of runtime in the Q-learner the state space had to be limited. The state parameters were chosen to be only On Balance Volume, Price / SMA and Bollinger Bands. The choice to not include holdings in the state further complicated the wrapper (StrategyLearner.py). However, by reducing the state space for the Q-learner, runtime was improved over the alternative.

Thresholds for the discretization of each of the indicators were determined experimentally, several reasonable combinations of thresholds were tried and the best performing across multiple assets, primarily JPM, IBM, AAPL and DUK, were chosen for the final learner.

On Balance Volume was given ten possible states (0-9). The usual trading volume for separate assets can vary wildly, to accommodate this the spread of the ten states was based on a standard OBV. Standard OBV was determined as a sum of the first five days trading volume for the asset and time period in consideration. The threshold for a ‘9’ state was set to include any days with an OBV greater than standard OBV, a ‘0’ state for all days with an OBV less than -1 * Standard OBV. Intermediate thresholds were determined experimentally.

Price / SMA was also given ten possible states (0-9). The thresholds for all these states were determined experimentally. It was found that the most useful data to the learner was near the 1.0 price / SMA mark. Six of the ten possible states are set within 0.2 of this mark. Information further than 0.5 of this mark was considered extraneous data that would be more effectively judged by the Bollinger Bands indicator.

Bollinger Bands were only given three possible states; above, below and within. This choice was made to reduce the state space, within which the Q-learner must operate, to improve runtime. The price / SMA indicator could capture the necessary information within the Bollinger Bands .

Discretization was performed with cascading ‘if’ statements. The use of masks for improved speed was considered, but, was ultimately unnecessary to meet runtime constraints. The state that was fed into the Q-learner was a three-digit integer as calculated in the following pseudo code.

```
State = Bollinger * 100 + On_Balance_Volume * 10 + Price_Per_SMA
```

This left the learner with 300 possible states. Concluding that the Q-learner would not know whether it was holding a position, the actions were selected to be either short, long or hold nothing. The wrapper (StrategyLearner.py) would determine what trades needed to be made to move into the desired position returned by the Q-learner. The final Q-table would be a total of 900 cells (3 actions * 300 states).

The wrapper (StrategyLeaner.py) would for every epoch on every day closing feed the new state and reward into the Q-leaner and implement actions as detailed in the following pseudo code:

```
# In Sample Training
While Q-learner not converged:
      Initialize state, action, trades, holdings and portvalues
      For days in date range:
            If action is short and position not short:
                  Trade into a short position
            Else if action is hold nothing and holding a position:
                  Trade into all cash
            Else if action is long and position not long:
                  Trade into a long position
            Else:
                  Do nothing
            State_prime = Tomorrow’s state
            Reward = Tomorrow’s daily return * holdings – /
            commission and impact
            Action = query_Q_Learner(state_prime, reward)

# Out of sample Testing (no learning)
Initialize state, action, trades, holdings and portvalues
For days in date range:
      If action is short and position not short:
            Trade into a short position
      Else if action is hold nothing and holding a position:
            Trade into all cash
      Else if action is long and position not long:
            Trade into a long position
      Else:
            Do nothing
```

```
State_prime = Tomorrow's state
Action = query_set_state_Q_Learner(state_prime)
```

REWARD FUNCTION

The reward function was simply chosen to be the daily returns on the assets times the number of stocks held (short or long) minus the effects of commission and impact. The Q-learner received a reward every day to speed up convergence.

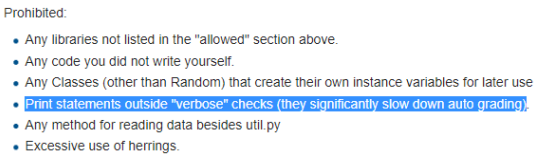
CONVERGENCE

Originally convergence was simply assumed to require ~ 500 epochs, this proved overly conservative and time consuming. Implementing a similarity-based convergence with a minimum number of epochs lead to the program finishing in ~25 epochs. This single modification reduced total runtime from ~25 seconds to ~4 seconds, for the learning stage. The logic for this convergence is provided in the following pseudo code:

```
WHILE #epochs < 10 OR Portfolio value of last epoch more than
0.1% different from Portval of this epoch
     continue in sample training
```

PROGRAM NOTES

As a note to the grader I find it necessary to mention that due to fear of the following part of the grading rubric:

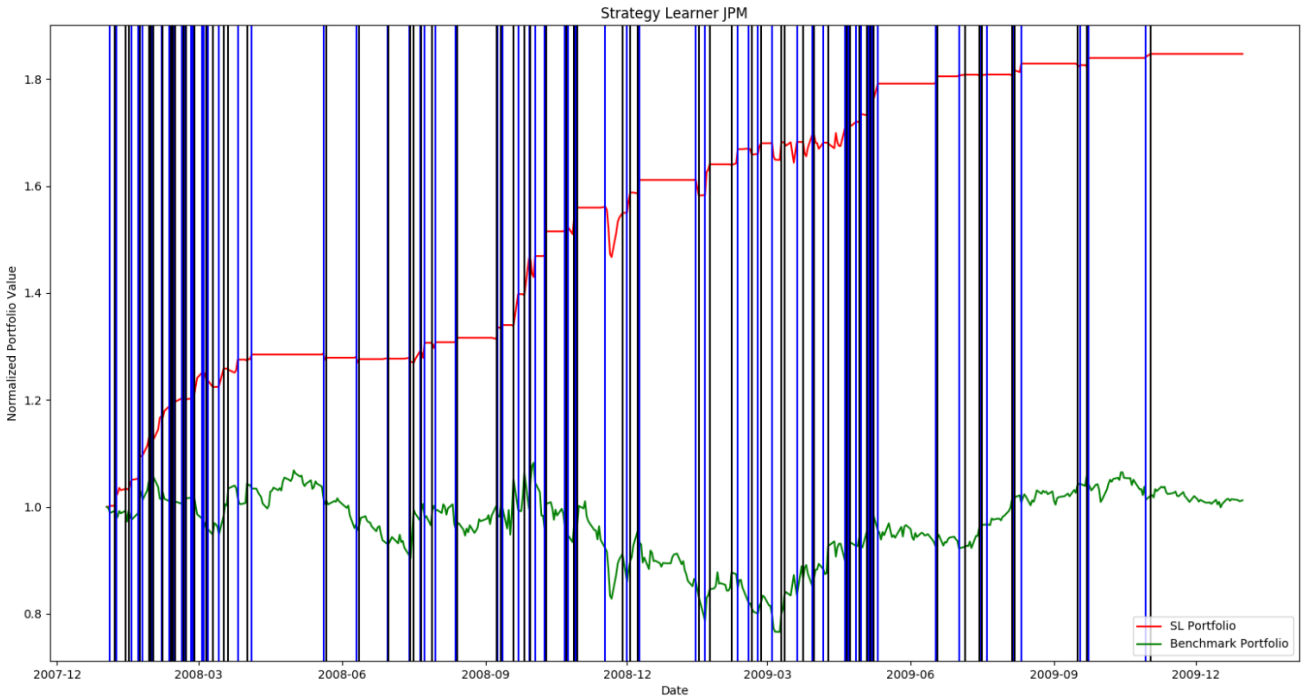

Prohibited:

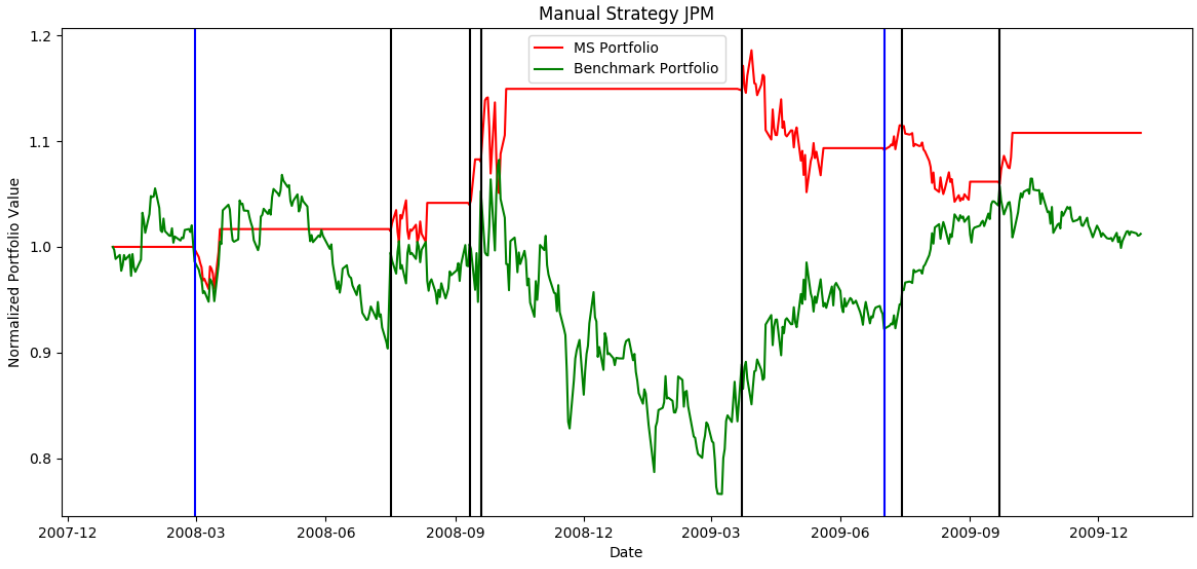
- Any libraries not listed in the "allowed" section above.
- Any code you did not write yourself.
- Any Classes (other than Random) that create their own instance variables for later use
- Print statements outside "verbose" checks (they significantly slow down auto grading).
- Any method for reading data besides util.py
- Excessive use of herrings.

StrategyLearner.py, experiment1.py, experiment2.py, indicators.py, and ManualStrategy.py have been written to default to no output. If you run these programs directly (using *if __name__=="__main__":*) you will have to set verbose to True in the "*learner = StrategyLearner(verbose = False, impact = 0.000)*" line to print statistics and plots. In experiment1.py and ManualStrategy.py you must also set verbose to True in the line "*MS = ms.ManualStrategy(verbose=False, impact=0.005)*". In indicators.py you set verbose to true in the line under "*def testcode()*" " *portvals = compute_portvals(verbose=False , start_val=sv)*" if you wish to see output.

Experiment 1

COMPARISON TO MANUAL STRATEGY





The above figures compare a benchmark portfolio (JP Morgan 1000 shares long) to the portfolios created by the Strategy Learner and the Manual Strategy from project 6. Blue vertical

lines represent entering 1000 shares long positions and black, 1000 short. Impact is set to 0.005 in both strategies. These results are generated from in sample testing. It is instantly noticeable that the Strategy Learner came up with a much better trading strategy than I could on my own with the Manual Strategy. The Strategy Learner also found order of magnitude more trading opportunities than I could manually.

StrategyLearner.py and experiment1.py are nearly the same program. The only difference being in the asset considered (JPM instead of AAPL) and the incorporation of a call to ManualStrategy.py. Description of assumptions made and parameters utilized in this experiment are in the above sections detailing the Strategy Learner. All testing is IN SAMPLE.

The Strategy Learner greatly outperformed the Manual Strategy as detailed in the following statistics:

```
Strategy Learner
0.6434665 cumulative_return_SL
0.0123 cumulative_return_benchmark
[ 0.00099586] mean_daily_returns_SL.values
[ 0.0013929] mean_daily_returns_benchmark.values
[ 0.00486877] std_daily_returns_SL.values
[ 0.05209278] std_daily_returns_benchmark.values

Manual Strategy
0.107889 cumulative_return_MS
0.0123 cumulative_return_benchmark
[ 0.00024112] mean_daily_returns_MS.values
[ 0.0013929] mean_daily_returns_benchmark.values
[ 0.00877764] std_daily_returns_MS.values
[ 0.05209278] std_daily_returns_benchmark.values
```

This staggering difference in performance should repeat itself on every asset in every time frame assuming you continue to test in sample. The Q-learner can tailor itself to any situation whereas the rules of the Manual Strategy are rigid and unchanging. The only way the Manual Strategy could possibly outperform the Strategy Learner on multiple assets is if it's rules were manually rewritten for each asset and time period (by a more skilled investor than I).

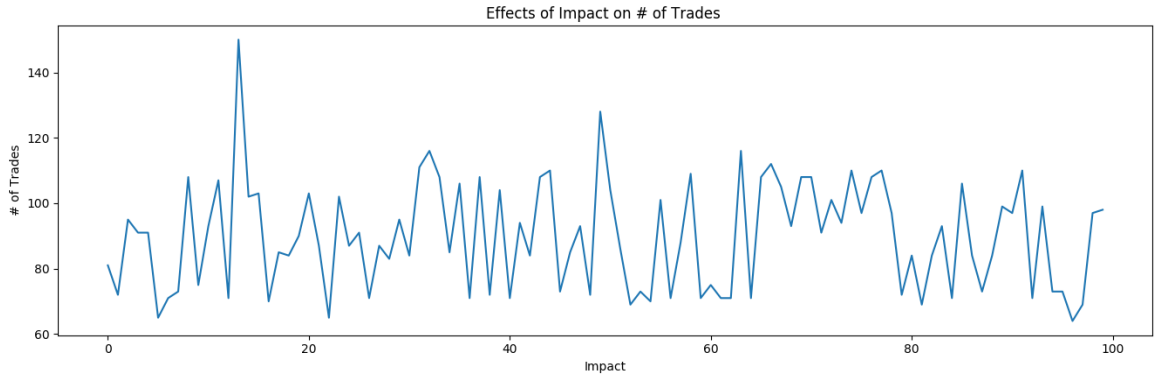
Experiment 2

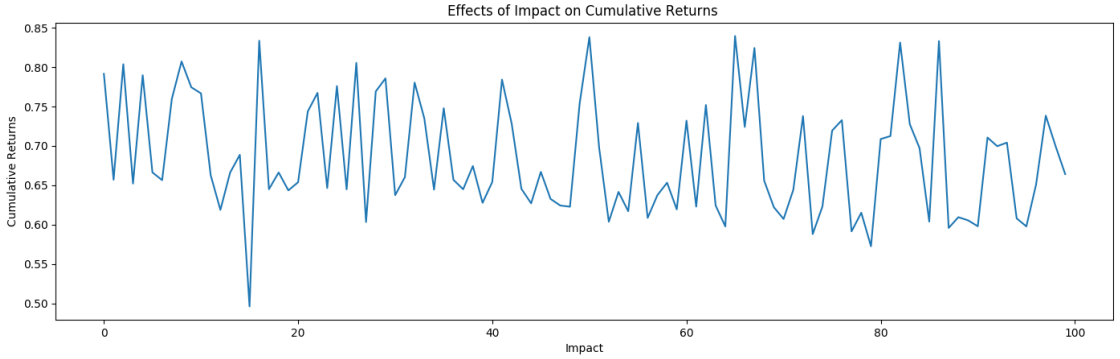
EFFECTS OF IMPACT

Hypothesis

Impact always punishes the trader for every trade he makes, the more trades, and the higher volume of trades will increase this punishment. To implement this effect in Q-learner reduced rewards were received based on the impact. I hypothesize that, due to this punishment for

each trade, the learner should make fewer trades AND less money as impact consumes more capital. Graphs follow detailing the results of running by strategy learner 100 times on incrementally increased levels of impact. X axis impact values are multiplied by 1000 for easy reading.





Conclusion

My hypothesis was … bad. It is clear from the above graphs that, as we introduce greater and greater amounts of punishment for each trade, the Q-learner is affected randomly. I was wholly incorrect in believing that this punishment would reliably reduce cumulative returns as well. I contribute this effect to a lack of a holdings state. The Q-learner has no idea what it is holding and assumes all necessary information about previous states is included in the current state, which in the case of impact is untrue. Without this information the Q-learner sees punishments from impact as random, and in turn, responds randomly.

This issue could be resolved by implementing holdings as a state. This change would likely be worth doing despite the increased runtime necessary for a larger Q-table. However, I have chosen to submit my project as is because I believe there is value in this lesson.

As an experiment I changed the code in experiment2.py to explicitly not punish a choice to not trade. This had no effect, reinforcing my conclusion that the best way to get the Q-learner to respond to impact would be to add holdings to the state.

Program notes

The program experiment2.py simply runs the code from StrategyLearner.py repeatedly with different levels of impact. No other modifications were made. This program takes a LONG time to run ~ 10 minutes. Additionally to display the included plots you will need to change line *"verb = False "* in *"if __name__=="__main__":"* to "*True*".