Final Project : strategy_learner

CS 7646 Machine Learning for Trading

Description of the Strategy Learner

Framing the problem

Defining the QLearner's needs and outputs

My final strategy learner learns from the reward, which is defined by the change in the value of the portfolio on a given day due to changes in stock pricing and the cash position. It takes in the previous day's reward information and the current position (LONG, SHORT, CASH) and the state, which is calculated using the technical indicators for a given day. Note: I describe below the way I approached the thresholds and discretization of the data to create specific buckets for the state. The QLearner provides an action signal by querying the Q-table looking at the state and the reward. As it is learning (during addEvidence), I maintained the QLearner's appetite for exploration using the alpha (0.2) and gamma (0.9) values from the QLearning robot, as well as a nonzero random action rate (rar).

Taking a Positional approach

Rather than going with an approach where the action recommended by the QLearner was used directly to determine the next step for the trader (ie: +1000, -1000), I used a blended approach where the signal from the QLearner (the recommended action) was converted into a position in an orders series. Position was calculated directly from action (action - 1), where a position of 0.0 is CASH, 1.0 is LONG, and a position of -1.0 is SHORT). I used these positions to build out an orders sheet before creating a trades dataframe. This approach allowed me to build out trades and holdings at the same time, and I was able to query both the prior and current positions, as well as the the existing holdings and previous trades, before deciding to write a trade action into the trades dataframe. This approach eliminated the possibility of getting into a holding position where I owned more than 1000 shares at a time, and it also streamlined the creation of my trades and holdings data.

Data adjustments

Basic adjustments. I made a number of adjustments to the data, starting with some typical nadrops and forward + backfilling techniques. I did not normalize the technical indicators for the learning phase, though they are normalized in the context of this report for visual simplicity.

Time Windows for rolling statistics was a variable that I played around with quite a bit. I started all of the rolling statistics at windows of 14 days, but quickly realized that this limited the number of trades I could make in the year, which limited my QLearner from exploring appropriately and learning an optimal policy. My final state for time windows are described below, alongside the individual indicators.

Thresholds. When I needed to discretize the possible states for the QLearner, I started by defining thresholds for the technical indicators. I did this by creating a dataframe of the daily values of the four technical indicators. Iterating over each indicator, I sorted the dataframe and set the thresholds using a

simple calculation of step size based on the specified number of steps from the constructor and the size of the data in hand.

Discretization was a bit more tricky, since I took the approach of having semi-continuous values for the state. To enable this state, I increased the default number of states for my QLearner to allow me to have a semi-continuous number of distinct states for the QLearner. To calculate semi-continuous values of state, I set the state value while iterating over all four of the technical indicators separately, determining the threshold for each, then applying an exponential formula to the result. This gives my QLearner the ability to explore and support thousands of states. I feel this would be useful in the case that we need to look at many years of data or at a portfolio containing multiple equities.

Leveraging convergence to meet time requirements and achieve eco-friendly compute use
I employed a convergence strategy to ensure that I met the time requirements for the learning phase (25 seconds), which also conserves compute resources and a tiny bit of energy. I implemented a small helper function that looked for convergence of the portfolio value from the learner. If the learner found the same outcome 10 times in a row and the epoch was more than 20 total epochs, I broke the loop and declared optimization at that point.

Chosen Indicators

Momentum
Given that I was looking over a small time window of about one to two years and using a reinforcement learner, I decided that using higher frequency indicators would provide a strong daily signal. Momentum measures the rate of the change of a stock's price, and I found that it is a useful indicator of strength or weakness in the current price.

- Formula:
  - momentum[t] = (price[t]/price[t-window]) - 1
- Window used: 4 days. Originally, I started with 14 days and started to see that my learner was struggling to effectively explore and create a healthy q-table and thus trading policy. I changed the window to 4 days to encourage more exploration, and as a result both in training and in out-of-sample testing, I saw an increase in the number of trades made (when no market impact was factored into the reward, ie: impact == 0.00)

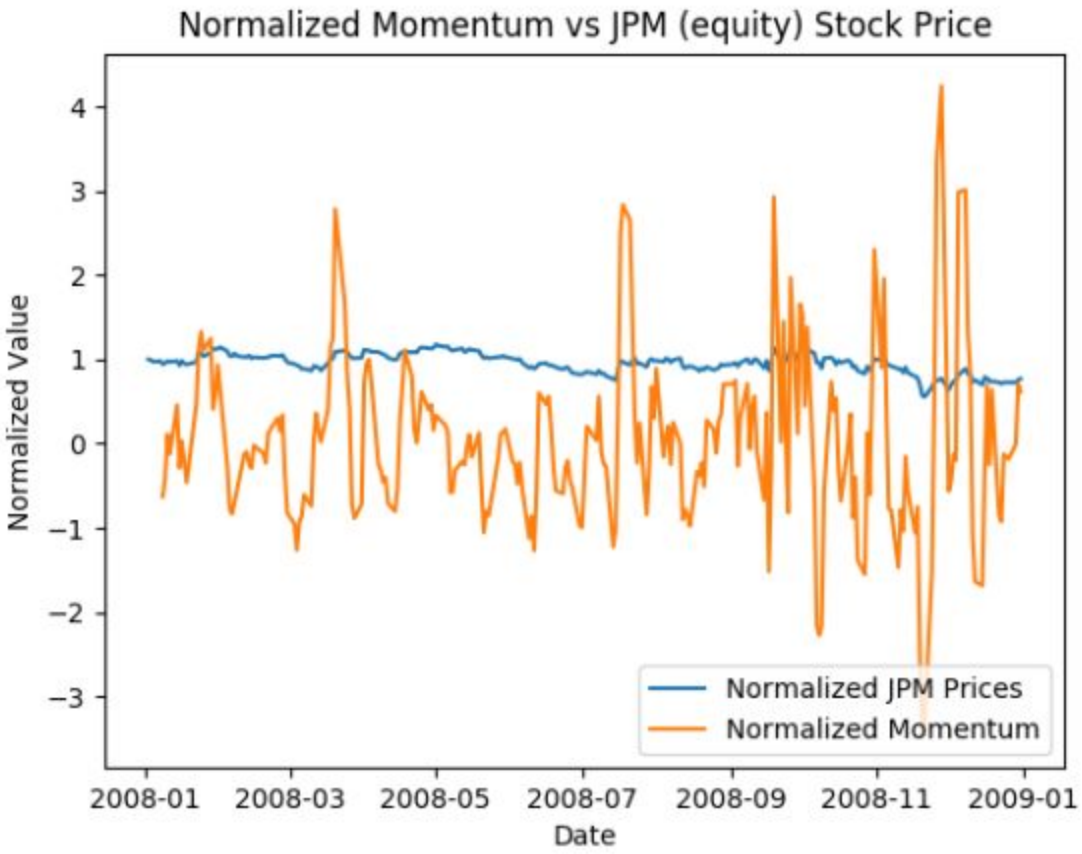

Figure: normalized momentum versus the value of JPM from Jan 2008 to Dec 2008

Simple Moving Average (SMA)

I know that most of my classmates chose price/SMA, but I started with SMA and saw great performance of my learned portfolio versus the benchmark, so I kept it. SMA is commonly used to identify support and resistance prices of stocks in the market, to identify trading opportunities. I felt that SMA was an ideal indicator for the situation of this project because we were provided with so much backward looking data.

- Formulas:
  - rolling_mean = prices.rolling(window=window).mean()
  - sma = (prices / rolling_mean) - 1
- Window used: 4 days. Similarly with momentum, I started with 14 days and then moved to 4 days to encourage exploration, though unlike momentum I suspect that shortening the window on SMA has a more significant impact on knee-jerk trading. If I were building this learner for personal use, I would likely move to a longer time window, closer to 8 to 10 days.

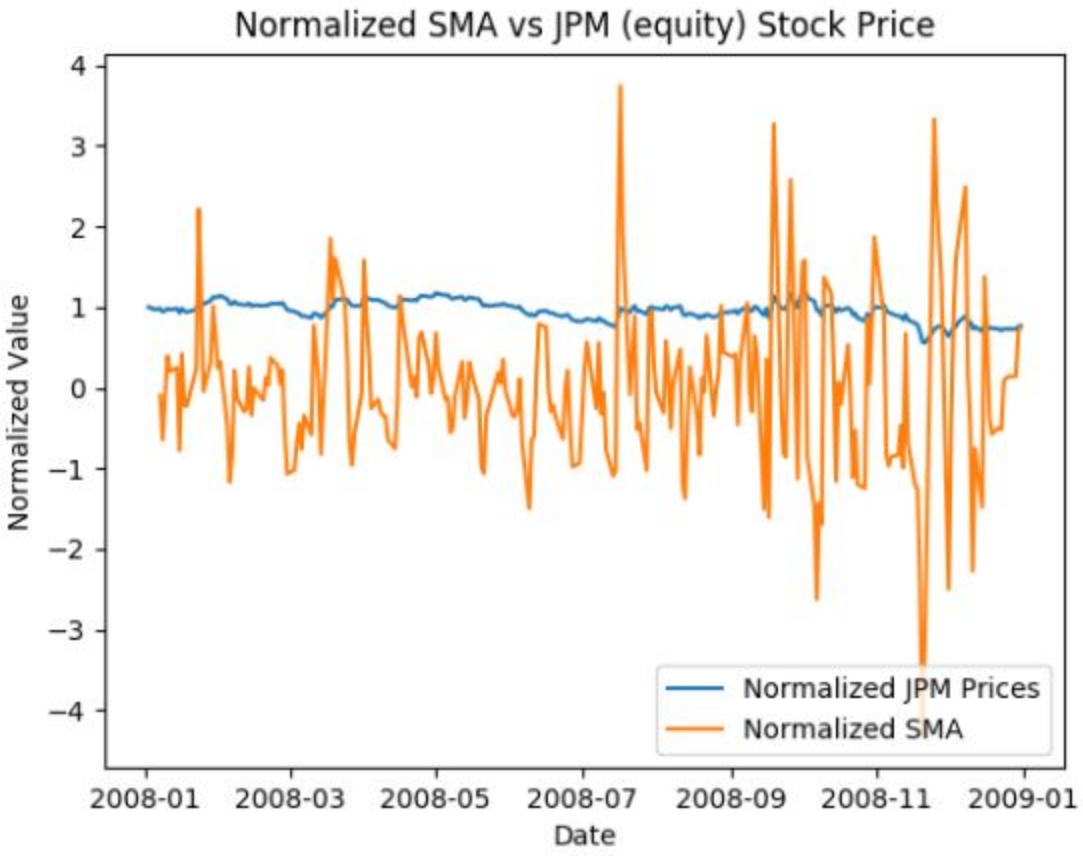

Figure: normalized SMA versus the value of JPM from Jan 2008 to Dec 2008

Bollinger Value ®

This technical indicator is one of the most popular for use in technical trading, and the author was intrigued to learn about John Bollinger's set of 22 rules to follow when using the bands as a trading system. When I learned of the vastness of the use cases, it felt like the perfect indicator to throw at the QLearner to see what it would do with it.

- Formulas:
  - rolling_mean = prices.rolling(window=window).mean()
  - rolling_std = prices.rolling(window=window).std()
  - bollinger_value = (prices - rolling_mean) / rolling_std
- Window used: 4 days. Similarly with momentum, I started with 14 days and then moved to 4 days to encourage exploration for my reinforcement learner, and like momentum I would personally feel comfortable with a trading strategy that looked on such a short window for this indicator. Like momentum, Bollinger Bands ® and Bollinger Values ® seek to identify areas where price is trending stronger or weaker than its natural value, thus I would see it as being good at identifying opportunities for arbitrage trading.

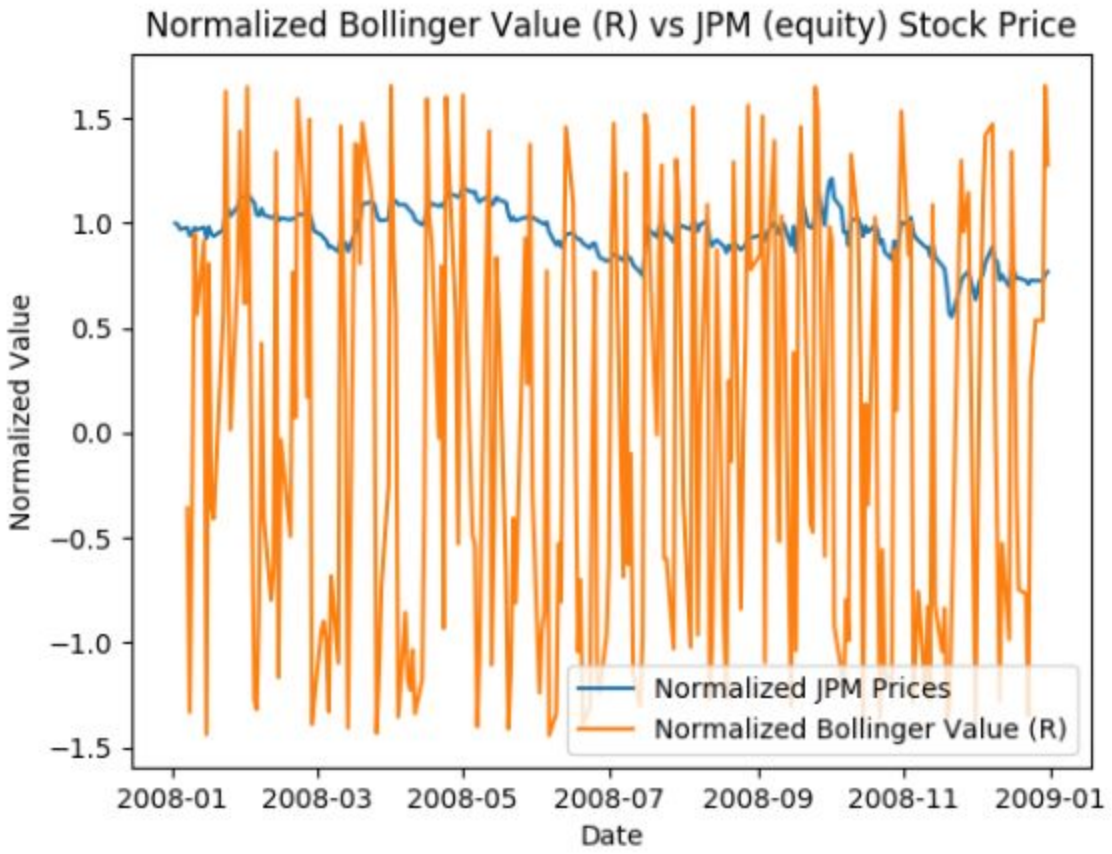

Figure: normalized Bollinger Value ® versus the value of JPM from Jan 2008 to Dec 2008. You can see how reactive this indicator is.

Volatility

This technical indicator is the author's favorite. Volatility, like beta, refers to the amount of uncertainty or risk related to the size of changes in a security's value. Higher volatility values means that the price of the security can fluctuate dramatically over a short time period, whereas lower volatility means that a security's valued tends to be more steady. Given that I selected two arbitrage-identifying indicators (momentum and Bollinger Value ®) as well as SMA which I sense is more useful in determining longer positions, volatility felt like a fantastic indicator to round out the lot for my QLearner.

- Formulas:
  - rolling_std = prices.rolling(window=window).std()
  - volatility = rolling_std
- Window used: 4 days. Similarly with momentum, I started with 14 days and then moved to 4 days to encourage exploration, though like SMA or perhaps even more acutely so than in the case of SMA, I suspect that shortening the window below 6 or 8 days weakens the value of the indicator. If I were building this learner for personal use, I would likely move to a longer time window, closer to 14 days.

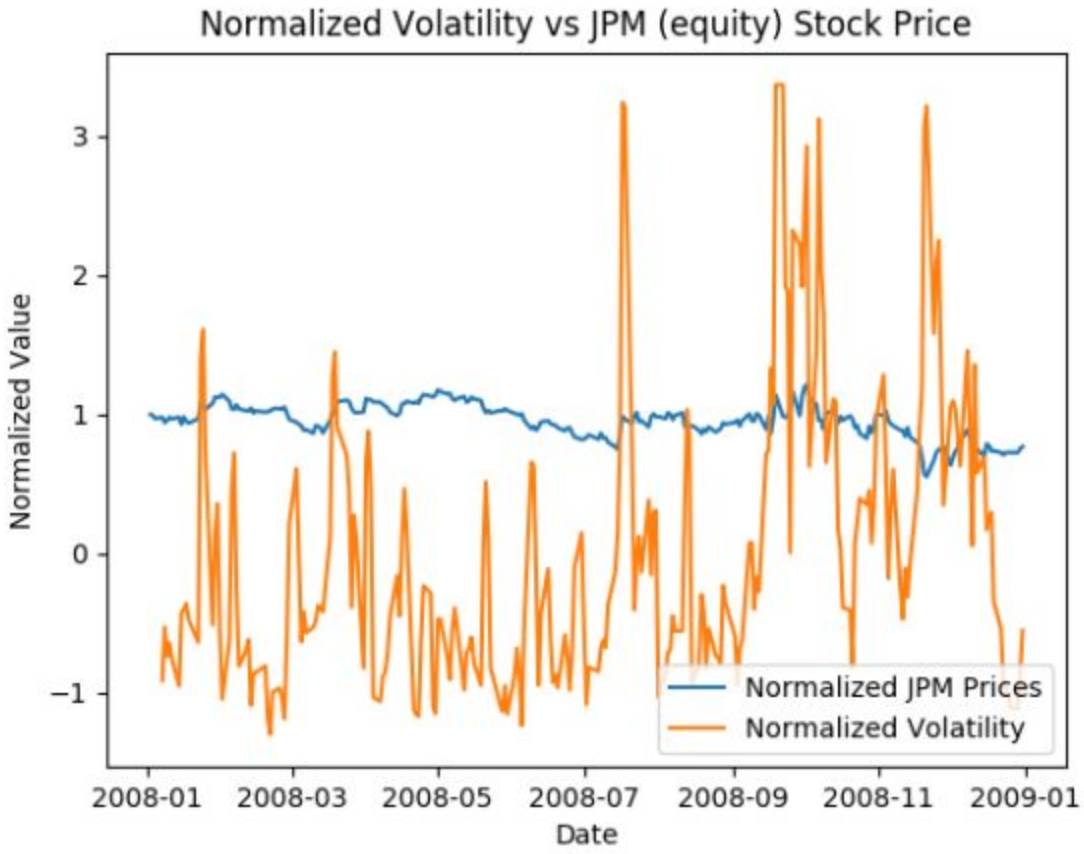

Figure: normalized volatility versus the value of JPM from Jan 2008 to Dec 2008.

Experiment 1

Experimental design

Plotted the in-sample normalized returns (a measure of performance) of the benchmark versus the learned strategy for the date range specified in the benchmark, to see how well the learned portfolio performed versus the benchmark portfolio.

The benchmark is a portfolio starting with $100,000 cash, investing in 1000 shares of the symbol in use ("JPM") and holding that position until the last day in the date range. Requirements:

- Sell the position on the last day.
- Include transaction costs.
- In sample/development period is January 1, 2008 to December 31 2009.
- Out of sample/testing period is January 1, 2010 to December 31 2011.

The code that implements this experiment and generates the relevant charts and data can be found in experiment1.py

Assumptions and Parameters

- Assume that the technical indicators chosen for the StrategyLearner will work in this date range, that JPM was trading on those days, and that both the benchmark and learner would experience enough change to demonstrate concepts and learnings.
- Parameter values:
    - start_val = 100000,
    - benchmarkSymbol = "JPM",
    - commission = 0.00,
    - **impact = 0.00**,
    - num_shares = 1000,
    - sd=dt.datetime(2008, 1, 1),
    - ed=dt.datetime(2009, 12, 31)

Output Graph: Portfolio Returns

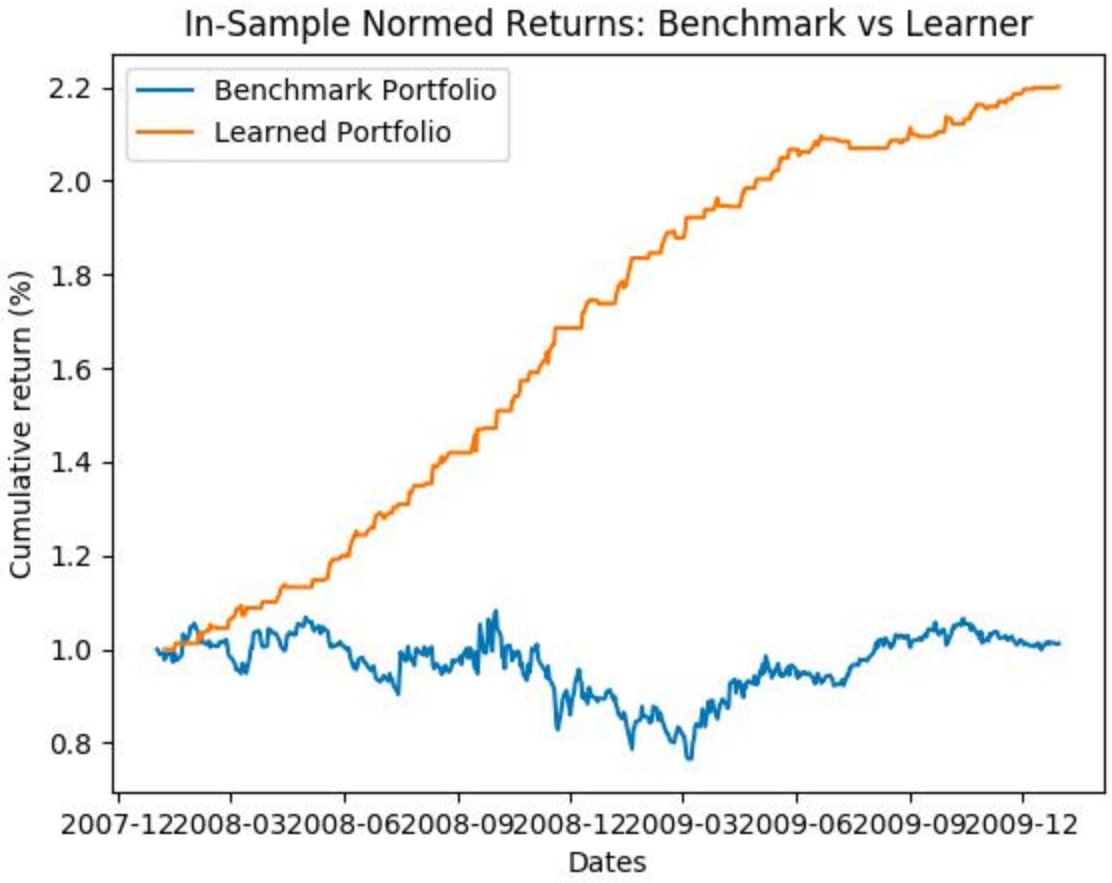

Figure: normalized (to first day price) benchmark portfolio returns of JPM versus StrategyLearner portfolio, from Jan 2008 to Dec 2008, using **in sample data**.

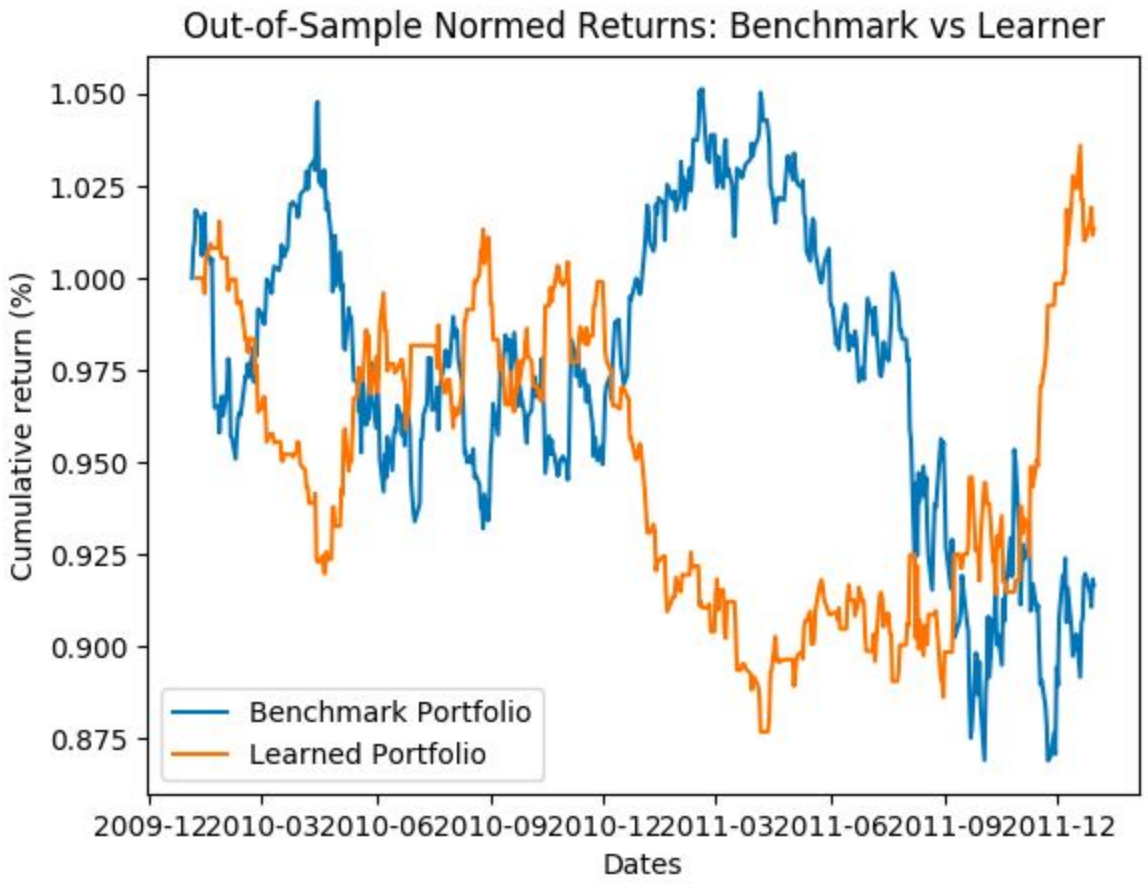

Figure: normalized (to first day price) benchmark portfolio returns of JPM versus StrategyLearner portfolio, using **out of sample data**.

Learnings

The learner performed significantly better than the benchmark portfolio in-sample, as expected. It's hard to imagine a scenario where this wouldn't be expected, given that it's all hindsight and the benchmark portfolio is tied to an immobile strategy.

However, out-of-sample is where things get more interesting. The portfolio from the StrategyLearner appears to do exceptionally poorly when a LONG position is desirable, and very well when a SHORT position is desirable, as evidenced by the mirror-imaging of the graphs at multiple points along the graph. Since we know that the benchmark is holding LONG, we can tell that the learned portfolio performs oppositely. The other interesting thing to note is how the learned portfolio really rallies at the end of the year. If you were to only look at the returns through mid-2010, you might not be impressed. However, when looking through end of December 2011, you see the portfolios once again diverge, this time with the StrategyLearner portfolio out ahead, with returns where the benchmark sees losses.

This, combined with the individual analysis of the indicators from above tells me that I would need to better optimize my learner, perhaps looking at other indicators like RSI or Price/SMA to see if there are indicators that would increase my returns in a less volatile manner.

Experiment 2

Hypothesis

Unless the learner can learn a strategy to mitigate the market impact of trading, increasing the value of market impact will reduce the cumulative returns for the learned portfolio (versus market impact of 0.00). Adding in market impact to pricing may also lower the total number of trades that the learner makes, since the reward will be less for each trade. My hypothesis is that the case where market impact is greater than 0.00 will likely present more "hold" signals than in the impact = 0.00 case from Experiment 1.

Experimental Information

The code that implements this experiment and generates the relevant charts and data should be submitted as experiment2.py. Many of the assumptions from Experiment1 are valid for this experiment as well. See above.

Parameter values:

- start_val = 100000,
- benchmarkSymbol = "JPM",
- commission = 0.00,
- **impact = 0.005**,
- num_shares = 1000,
- sd=dt.datetime(2008, 1, 1),
- ed=dt.datetime(2009, 12, 31)

Output Graph: Portfolio Returns

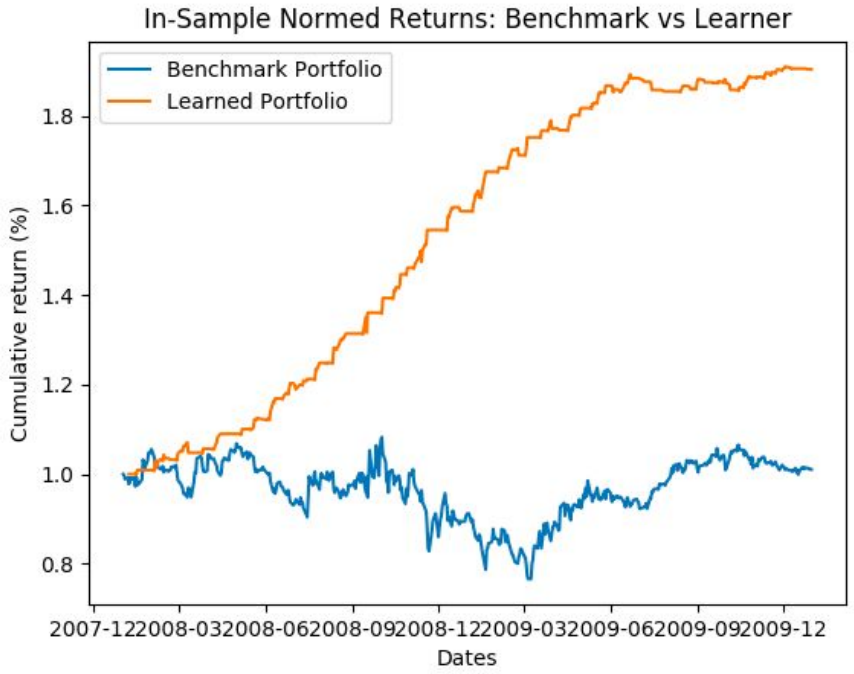

Figure: normalized (to first day price) benchmark portfolio returns of JPM versus StrategyLearner portfolio, using **in sample data** and a **market impact of 0.005** (.5% per trade).

Number of trades (in sample)

Query: `learnerTradesDF.loc[learnerTradesDF['JPM'] != 0]`

When market impact = 0.00

- Benchmark: 2 trades
- Learner: 158 trades in ~500 trading days

When market impact = 0.005

- Benchmark: 2 trades
- Learner: 168 trades in ~500 trading days

Learnings

The outcomes appear to agree with half of the hypotheses generated by the author. Outcomes suggest that in sample, cumulative portfolio returns are less for the StrategyLearner portfolio with market impact (~1.8 in sample with 0.005 impact vs 2.2 in sample with 0.00 impact).

On the flip side, however, the number of trades for the learner with market impact is higher (by 10 trades over the same time period) than for the learner with no market impact. The author humbly offers two explanations: (1) my code could be incorrect, or (2) the learner still makes net positive money with each trade, so it tries to make more trades despite rewards lowered by market impact, to achieve the same or higher cumulative returns.
