Building a Trading Strategy using Machine Learning

Overview

In this project, I use machine learning to build a trading strategy, using historical price data of JPM stock between Jan 1 2008 to Dec 31st 2009. This strategy is then compared and evaluated on 1) out of sample data from Jan 1 2010 to Dec 31st 2011 on JPM stock 2) against a manual rules based trading strategy built earlier and 3) with different values of impact

Choosing the Predictor Features (Indicators)

To build a strategy learner, I first devised a series of technical indicators using JPM stock price data during the in sample training period that were used for training purposes by the Random Forest algorithm.

These indicators included:

1) **Price/SMA %** = Price/20 Day Price Moving Average - 1 (which tells us how much above or below the price is above its simple moving average)
2) **Volatility** = Standard Deviation of Daily Price Changes in the last 20 days
3) **Bollinger Band % =** (Price - Lower BB Band Price)/(Upper BB Band Price - BB Lower Band Price) to tell us where the Price is relative to the upper and lower Bollinger Bands
4) **Momentum =** the % change in price over a period of N days in the past. Momentum = price[t]/price[t-n] - 1 where t = the day we are calculating momentum for
5) **MACD and Signal Line Crossovers =** Days when the MACD crossed above the signal line or when the MACD falls below the signal line are assigned a value of 1 else they are assigned a value of 0

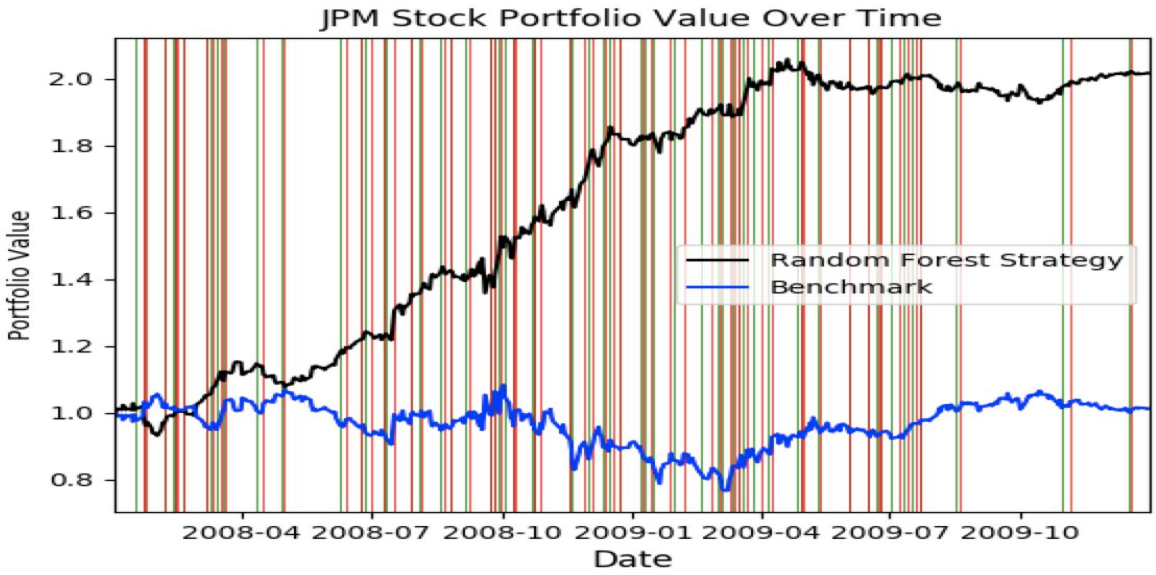
(Note: Since all these indicators are either % changes, standard deviations, or Binary indicators (1 or 0) they are already in a normalized form and further normalization is not necessary as we are not considering absolute price changes here)

I decided on these set of features after experimenting with a few more feature combinations that also included other indicators like MACD value, price sma crossovers, and bollinger band crossovers and noting their performance on the training data.

To evaluate performance of different technical indicator combinations, I set a fixed seed value (106347), then passed the feature combinations into a Random Forest Learner, and then evaluated the cumulative return of the trading strategy using the Random Forest predictions on the in sample data with only those set of feature indicators. The combination above gave the best cumulative return among the few combinations I tried (Note: However, I only experimented with a few subsets of features. To find the optimal best possible combination, we would need to do a brute force grid search across every single combination. This would have taken a lot of time so I settled on just a few random combinations)

The in sample performance of using the following features is shown below. (A benchmark where we buy JPM on the first day and hold) is plotted for comparison purposes. Red Lines indicate Short Entry points. Green Lines indicate Long Entry Points below and in the rest of the charts).

Figure 1- JPM Stock Portfolio Value using Technical Indicators for Strategy Learner



Choosing the Random Forest Hyper-Parameters

The two parameters for the Random Forest learner, number of bags as well as the minimum leaf size were identified after training on in sample data from Jan 1 2008 and Dec 31st 2008 and then noting their performance on validation data from Jan 1 2009 to Dec 31st 2009. In the end 25 bags were chosen, and the leaf size was set to 5.

This is because the gains in cumulative return after 25 bags were negligible maxing out at ~$205K and not increasing further by much even when bags were set to 30,40 and 50. Setting a leaf size of less than 5 led to overfitting as seen by degrading validation sample performance where cumulative returns increased on in sample performance but decreased on validation data for leaf sizes less than 5.

Choosing the Target Variable (The Predicted Action)

For learning purposes, instead of predicting the next N day numeric price change (a continuous variable) using the technical indicators above, the problem was reclassified into a multi-classification problem for simplicity. The predicted variable (instead of N day return) was converted into a categorical value where:

- If the next N day price change was > threshold value then it was classified as a 1.
- Else if the next N day price change was < negative threshold value then classified as -1
- Else 0

Each of these 3 predicted values (1, 0, -1) correspond to a logical action that we would want to take using our strategy learner. For example if the price increase was greater than a threshold value we would buy Long because we expect the price to increase over the next N days, if the decrease was less than a threshold value then we would buy Short as we expect the price to decrease or else do nothing. The reason we have a threshold value to make decisions is because by focusing on only meaningful price changes we could avoid noise coming from very small or uneventful price movements. To find the best values of N and Buy/Sell Thresholds, I had to do some searching:

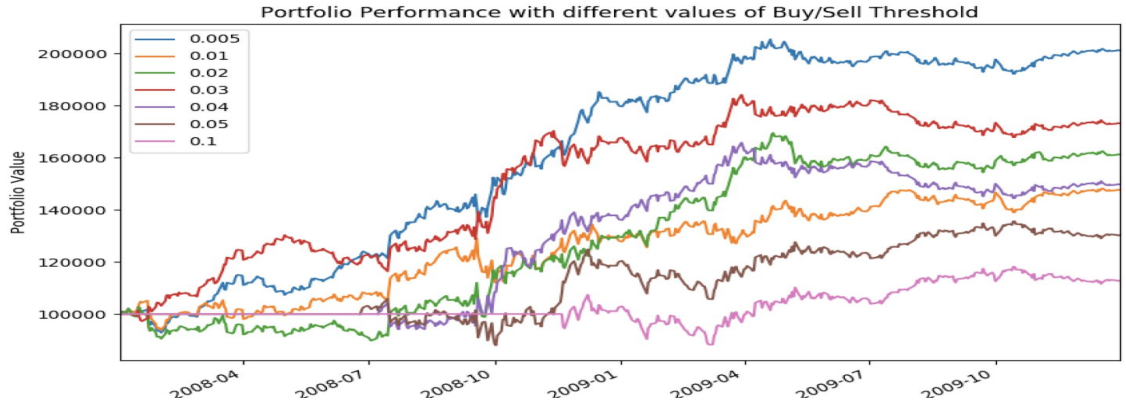
1) The threshold value itself to use to indicate a buy or a sell itself was determined by running the Random Forest Learner over different threshold values and noting the cumulative return for each
2) The N used for determining N day price change, based on which I determined an action was also determined using different values and noting cumulative return. (seed was constant)

Figure 2 - Table summarizing performance of trading strategy with different Y value hyperparameters

Buy/Sell Threshold	N for N day Price	In Sample Cumulative Return %
0.5%	1	67%
1%	1	63%
2%	1	110%
3%	1	88%
0.5%	3	93%
1%	3	77%
2%	3	77%
3	3	74%
0.5%	**5**	**102%**
1%	5	41%
2%	5	62%
3%	5	73%

From the above, the 2 best performing combinations are (2% threshold value, Next 1 Day Price Change) or (0.5% threshold value, Next 5 Day Price Change). I decided on the latter combination because the differences in performance are small and by looking into the future over a slightly longer time frame, we can smooth out rapid price fluctuations that are just daily noise and not indicative of an actual price trend. Therefore, we predicted as 1 or -1 if the price increased or decreased by at least 0.5% (or 0.005) after 5 days in the future. The chart below also plots the performance of the portfolio when N = 5 and the threshold is adjusted. The best performer is when Buy/Sell Threshold = 0.005. (Refer to **experiment3.py**)

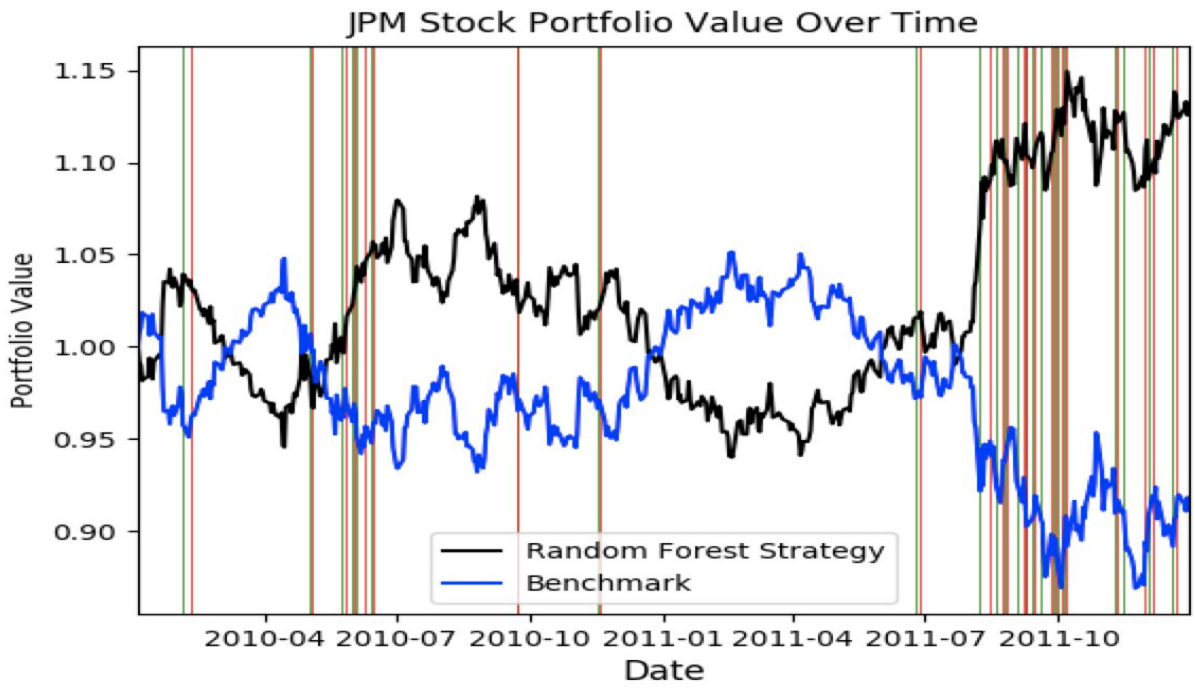
Figure 3 - Performance of Portfolio with different Buy/Sell Thresholds for next N day Returns(when N = 5)



Experiment 0 - Out of Sample Performance (Test) of Strategy Learner

With the technical indicators developed, random forest hyperparameters chosen, and the predicted variable identified I then trained a random forest on unseen test, out of sample data. The random strategy learner doesn't do as well, as expected due to overfitting on train data, but still manages a cumulative return of 15%.

Figure 4 - Out of Sample Performance (Test Period) for Random Forest Strategy Learner

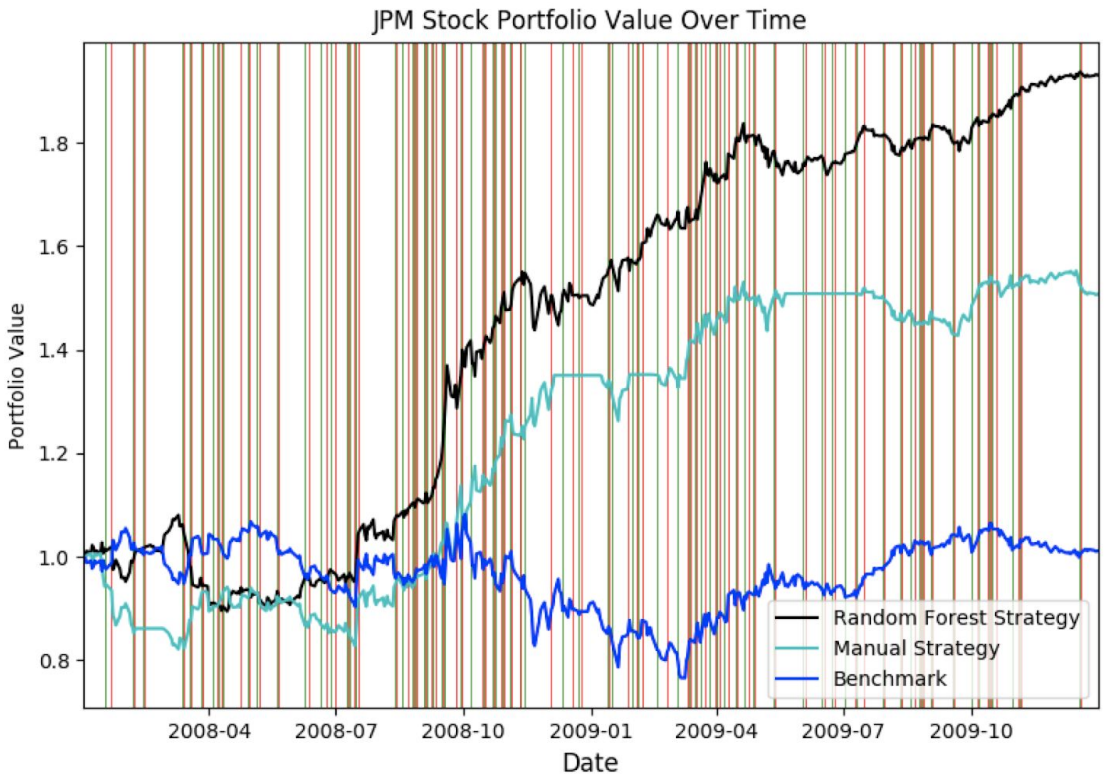


Experiment 1 - Comparing Performance of Manual Strategy vs Random Forest Strategy Learner

The random forest learner was now trained using the exact same indicators described in the Manual Strategy report but using the same parameters described above for the development of the Y indicator (5 Day Price Change, Threshold = 0.05%). The indicators for the new strategy were 1) Price/SMA %, Volatility Indicator, Bollinger Band %, MACD and Signal Line Crossovers and the Random Forest hyperparameters were Bags = 25, Minimum Leaf Size = 5. The technical indicators are exactly the same used in manual strategy. Performance is summarized below when seed = 106348.

Figure 5 - Table + Chart showing Performance of each strategy during in sample period

	Cumulative Return %	**Sharpe Ratio**	**Std Dev Daily Returns**
Random Forest Strategy	93%	1.85	0.011
Manual Strategy	51%	1.10	0.013
Benchmark	1%	0.15	0.017
Average Random Forest (10 Runs)	75%	1.62	0.011



From the above it can be seen that the random forest strategy beats both the manual strategy and the benchmark. It outperforms manual strategy, in sample, by 42% points and also has lower volatility as seen by a higher sharpe ratio and lower standard deviation.

However, because, the random forest learner has an element of randomness to it (each time a different tree can be built due to splitting on different and random indicators and indicator values), if we repeated this strategy multiple times we could see different results where the performance may be better or worse than the one shown above. Therefore for that reason, the above experiment was repeated 10 times with a different seed for the Random Forest, and each time cumulative return was noted. After 10 runs, cumulative return was averaged for all the runs. Even after multiple runs however, the random forest on average, performs better (75% cumulative return) than the manual strategy (51% cumulative return). Please refer to **experiment1.py** to simulate the experiment results.

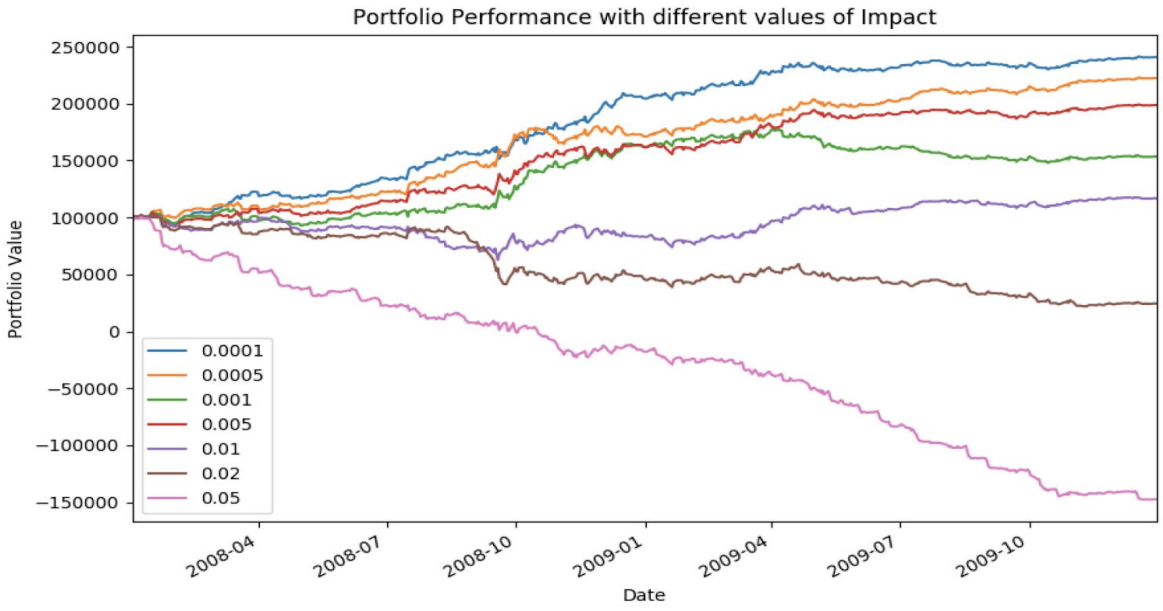
Experiment 2 - Effect of Impact on Portfolio Performance

Impact is how much we expect the price to change after we try to make an order. This is especially important if we are a very large investors with billions in funds because making large investments can cause upward/downward price pressure that can impact how much we actually end up paying after a trade. Because of this uncertainty, we expect prices to change a certain % when making trades which can affect the portfolio value and cumulative returns.

One hypothesis for how impact affects returns is that the higher the value of impact we expect then the lower the cumulative return is expected to be because at every trade we lose a greater % of our cash to impact price changes. When impact is very large (let's say 5%) this can overwhelm any gains we might

have made from a trade as we would have to have prices increases by a much larger amount to recover that lost cash. To test this hypothesis I ran an experiment where the same random forest strategy learner (bags =25, leaf size = 5) was trained over the same seed and same technical indicators and Y predictors as before. Then we ran our learner over the training time frame multiple times making trades but with only different values of Impact each time (everything else held constant). The portfolio value over time is shown below for each run with different values of impact.

Figure 6 - Portfolio Performance using RF Strategy Learner with different values of Impact



From the chart above it can be seen that our hypothesis is correct and as the value of impact increases, the cumulative returns decrease too. For example cumulative returns are highest when Impact = 0.0001 (or 0.001%) at 141% but decrease to -250% when Impact is set to 0.05 or 5%. Please also refer to **experiment2.py** to simulate results

(Additional) Experiment - Performance of Random Forest Strategy on New Unseen Stocks (AAPL, GOOG, XOM, IBM)

So far we have only been looking at JPM stock. I built my strategy only using JPM and also validated and tested it on different time periods using only JPM. However, the logical question then is what if we wanted to trade in more than just JPM stock? How well does our strategy generalize across different stocks that we have never seen before? If we assume technical indicators like MACD, Price/SMA %, BB % generalize across stocks, then we should also hope that for our random forest strategy to be good then it should be able to generalize to new and unknown stocks. By looking at different stocks we can also see how much we overfit our strategy on JPM price data and not so much on true price stock trends.

Hence for this experiment I ran the Strategy Learner over new stocks that we had never seen before: AAPL, GOOG, XOM and IBM. The results were as follows on out of sample data from Jan 1 2010 to December 31st 2011.

AAPL = 81% Cumulative Return for Strategy Learner vs 141% for Benchmark
XOM = -33% Cumulative Return for Strategy Learner vs 18% for Benchmark
GOOG = -531% Cumulative Return for Strategy Learner vs 19% for Benchmark
IBM = -118% Cumulative Return for Strategy Learner vs 55% for Benchmark

From the above it is apparent that while our strategy learner does well on JPM data both in and out of sample, it's performance across new and unseen stocks is significantly worse. Therefore this means that the strategy learner will have to learn from a lot of additional unseen data first on new stocks before it can generalize across the stock market.

Conclusion

In this project, I built a strategy learner using the Random Forest Algorithm (Bagged Learner built using multiple Random Trees) and then trained that learner on JPM stock data, searched for the best parameters and indicator combinations, in order to make stock price predictions and help inform a trading strategy.

To evaluate the strategy, it was then compared both in and out of sample to a benchmark strategy. It was also compared in sample to a Manual Rules Based Strategy that was built using only boolean logic using technical indicators. In both cases, the Random Forest strategy outperformed both Manual and Benchmark strategy on JPM data. Therefore the "Strategy Random Forest learner" is a better algorithm on JPM data. However, there remains a lot of room for improvement, as seen from the poor performance of the Random Forest Learner on new and unseen stocks other than JPM. This tells us that the Random Forest Strategy Learner while it performs decently on data it has seen, nonetheless, is still not good at generalizing across the entire stock market and often even worse than a simple Benchmark strategy.