Written By : Anshuta Awasthi
GT user name : aawasthi32
Date : 20 -04-2019

Strategy Learner

Trading Problem:

To design the learning trading agent, I have made a choice to use **classification based strategy.**
Indicators used : Same indicators.py file has been used which was written and submitted in manual Strategy assignment. The indicators used are :

1. Bollinger Bands
2. Price by SMA
3. MACD Crossover (MACD - signal)

Strategy Learner Overview:

In strategy learner , we have applied the random forest algorithm i.e Bag Learner with Random Tree on the manually created dataset to learn when the stock should be traded long , short or do nothing.

Now for the creation of dataset , we have values of indicators like Bollinger bands , Price by SMA and MACD crossover serving as features . While the output (Y value) will be determined by N day future returns of the stock. Since I have opted for classification , I have further categorized the N day returns into -1 , 1 and 0 based on the threshold value. The reasoning for selected threshold is discussed in the steps below.

Steps to create StartegyLearner.py:

1. In Random Tree Learner and Bag Learner , replaced the “mean()” with “mode” from scipy library.
2. In the constructor of Strategy Learner, initialize the value of “N” , which is going to be used in calculation of future “N” day returns.
3. Initialize the bag learner instance with 20 bags of Random Tree and each Random tree has 5 leaf node.
4. Later we can adjust the values of “N” in N day future return and number of bags in Bag Learner and see for which values we are getting better results.
5. In the addEvidence method , we will create our training dataset and train the bag learner . As stated above ,the feature values of dataset will be values of the 3 indicators fetched from “indicators.py” by calling respective indicator function for the given in-sample dates.
6. The Y values of dataset will be based on N day future return of the stock. First we calculated the N day future return of stock for all the trading days and drop the last N rows which resulted in NaNs . Since I opted to use classification , the “N” day return further need to be classified as values -1 , 1 and 0 based on threshold.
7. The threshold values YBUY and YSELL , need to be chosen wisely so that they will give satisfactory result (better than benchmark) with most of the stocks as this strategy learner is not being optimized for one particular stock prices.
8. We need to take market impact also in account while deciding on YBUY and YSELL values.

9. I have chosen , not to hardcode the YBUY and YSELL values since that would not be generalized for other stocks .So the threshold values are calculated as below first:

```
Mean   =  mean of all "N day future return values"
Std    =  Standard deviation of "N day future return values"
YBUY = mean + std * 0.5
YSELL = mean - std* 0.5
```

10. Ytrain(i.e. Y values of training dataset) will be calculated as follows :

```
If  (N Day return - impact > YBUY):
    Ytrain = 1
Else if ( N Day return + impact < YSELL):
    Ytrain = -1
Else:
    Ytrain = 0
```

11. Since now we have X and Y dataframes of training (in -sample) data , we simply invoked the Bag Learner instance initiated in constructor on our training dataset.
12. Now we have our model trained , in testPolicy() method we will create the test dataset and query the model to retrieve the y values .
13. In testPolicy() , we created the Xtest dataset by invoking the indicator methods from "indicator.py".
14. Queried the bag learner with Xtest dataset to fetch the Y values indicating BUY , SELL or CASH as 1 , -1 and 0 values respectively.
15. Based on these y values ,"Trades" dataframe is created keeping the holdings constrained to 1000 (for BUY) , -1000(for SELL) and 0 (for CASH).The trade values are adjusted with allowable values 1000, -1000, 2000, -2000 and 0 .
16. The testPolicy() method finally returns the trades dataframe.

Data Adjustment : Following parameters were tweaked to get satisfactory in-sample and out-sample performance:

a). Value of "N " in calculating N day return : The value used is 10. Ran the code with different values and selected the one which gives optimum performance.
b). Leaf Size of Random tree : It was mentioned that leaf size should be set to 5 or higher to avoid overfitting. The value used is 5 in StrategyLearner.py code .Higher values were not giving good results.
c). Impact : Impact value used is 0 . How does performance gets affected by the change in impact values is discussed in detail in next section (for experiment 2).
d). No. of Bags in Bag Learner: Value used : 40
e). YBUY : Explained above at step 9
f). YSELL : Explained above at step 9

Experiment 1 :

The intent of experiment 1 is to compare the results of Manual Strategy we created in Project 6 and the Machine Learning strategy we created in this project and see which one gives better in-sample results.

Steps to perform experiment 1:

1. Initialized the symbol as “JPM” , start date and end dates as in-sample date ranges and start value as 100000.
2. Invoked testPolicy() function from **ManualStrategy.py** file to get the trades dataframe from Manual Strategy for “JPM” stock and in-sample date ranges.
3. Invoked addEvidence() and testPolicy() from **StrategyLearner.py** to fetch the trades dataframe from StrategyLearner. The value of impact used is 0.0
4. Created trades dataframe for **benchmark** by adding 1000 shares on first trading day and -1000 on last trading day.
5. Now we have 3 trades dataframe from Strategy Learner , Manual Strategy and Benchmark.
6. On invoking the modified **marketsimcode.py**(where it accepts trades dataframe as parameter) for all the 3 trades data frames respectively , we got the “portvals” of Manual Strategy , Strategy Learner and Benchmark.
7. Calculated the Cumulative Return, Average Daily Return,Standard Deviation and Sharpe Ratio for all of them.

How to run the code:

Inside strategy_learner directory execute the command :

```
PYTHONPATH=../:. python experiment1.py
```

Above command will display results and will generate the graph named **“Exp1.png”**

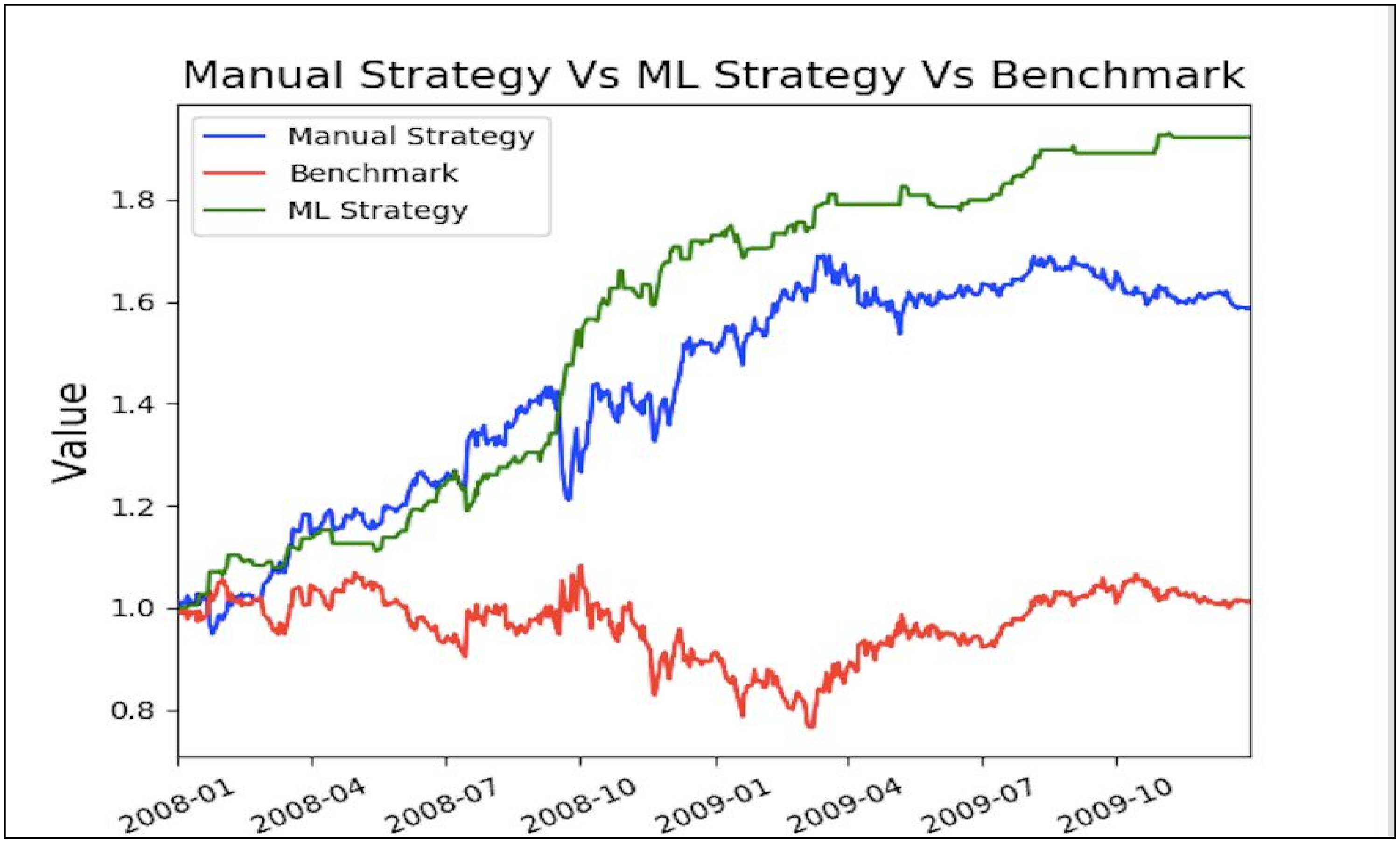
In-Sample Result Performance:

Date Range : 2008-01-01 to 2009-12-31
Symbol : JPM
Starting Value : 100000$
Commision : 0.0 $
Impact : 0.0

	Strategy Learner	Manual Strategy	Benchmark
Cumulative Return	0.9214	0.5871	0.0123
Std. Deviation of Daily Return	0.00697232167296	0.0116942295235	0.0170043662712
Mean of Daily Return	0.00132062942593	0.000985184384282	0.000168086978191
Sharpe Ratio	3.00679503812	1.3373533696	0.156918406424
Final Portfolio Value	192140.0	158710.0	101230.0

Below graph shows **normalized portfolio values** for Machine Learning Strategy , Manual Strategy and Benchmark:

Experiment1.py generated graph : Exp1.png



Analysis:

As above results and graph shows, that Machine Learning Strategy is performing better than Manual Strategy and way better than benchmark .Since Machine Learning algorithms can make better predictions than we humans do. As we see in graph , that machine learning strategy is only slightly better than manual strategy. However, it was seen that ML strategy will perform better than benchmark most of the time , but on repeated runs it was observed that sometimes ML Strategy **was NOT able to outperform** Manual Strategy many times.
The reasoning behind this is : In Manual Strategy , we tweaked our parameters to get best results for single stock "JPM" , while the Machine Learning Strategy (StrategyLearner.py) code has not been optimized for any single stock. It is the generalized one and idea behind this is **-** it should give good returns for most of the stocks , most of the time . However it is not guaranteed that this Machine Learning Strategy will give better results than Manual Strategy every time .

Would you expect this relative result every time with in-sample data? Explain why or why not.

Answer : No. Relative results were different

Above displayed results are the outcome when I have set the **random.seed(69)** in experiment1.py.

On removing the seed in experiment1.py and running it several times , it was seen that values of benchmark and Manual Strategy results were same every time , but the results were **different for Strategy Learner.**

Reason for this is , we are using Random tree based Bag Learner and in random tree the split attribute varies each time (selected randomly) even if we do not change the parameter values (like impact , N days , no. of bags , leaf size , YSELL , YBUY) in the Strategy Learner.

It was also observed that on removing seed value in experiment1 , many times it was giving results where Manual Strategy outperformed Machine Learning strategy. Reason being is in Strategy Learner code , we tweaked our parameters to optimize the results in a generalized way rather than for one particular stock (as we did in Manual Strategy).

Machine Learning Strategy developed in this project is better than Manual Strategy , since it is more generalized one and gives satisfactory results(Beats the benchmark most of the times) for many stocks.

Experiment 2:

The goal of this experiment is to observe how the changing values of *"impact"* affect the portfolio performance in *StrategyLearner.py* code.

Steps to perform experiment 2:

1. Initialized the symbol as "JPM" , start date and end dates as in-sample date ranges and start value as 100000.
2. Initialized the list of impact values with different values , starting with 0 and ending with 0.05 Impact values = [0 ,0.0005 ,0.0010, 0.0025, 0.005 ,0.010, 0.025, 0.05]

3. In a for loop ,invoked addEvidence() and testPolicy() from **StrategyLearner.py** to fetch the trades dataframe from StrategyLearner for different values of impact.
4. Calculated the Cumulative Return on each iteration.
5. Calculated the number of trades on each iteration.
6. Plotted the graphs for "No. of trades Vs Impact" and "Cumulative Return Vs Impact".

How to run the code:

Inside strategy_learner directory execute the command :

```
PYTHONPATH=../:. python experiment2.py
```

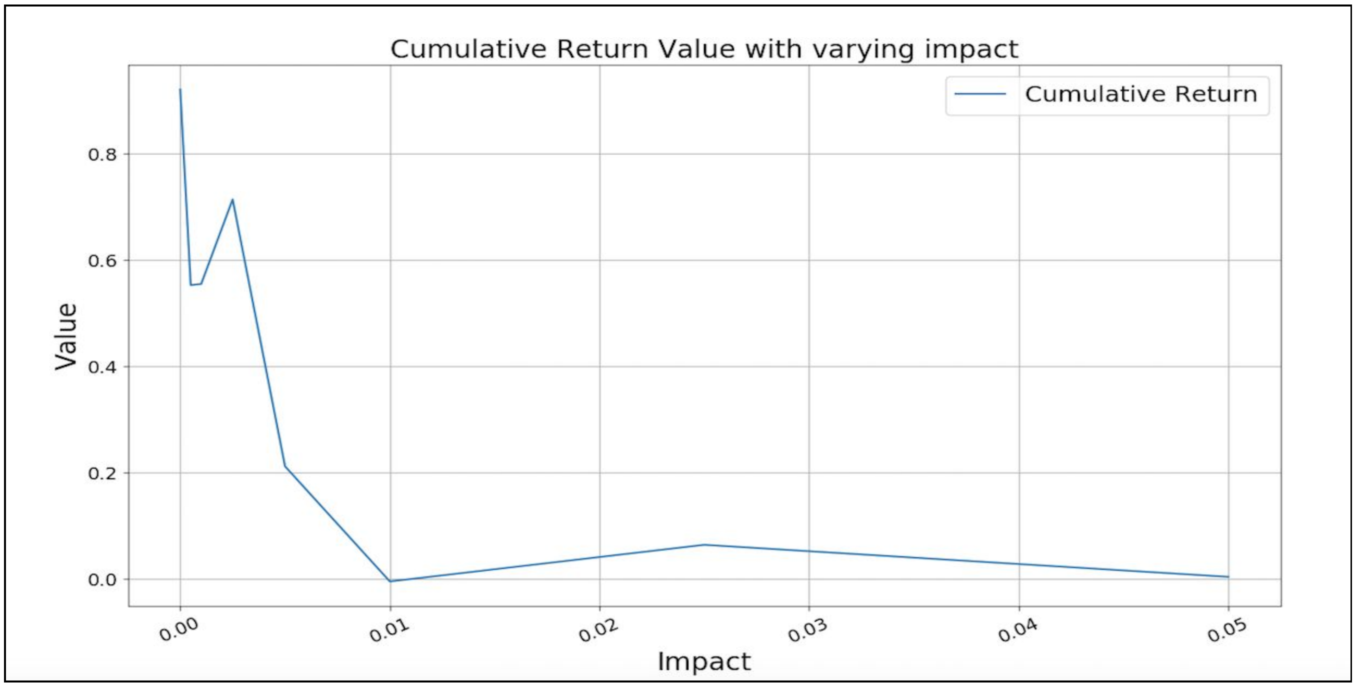
Above command will generate the graphs named **"Exp2_a.png" and Exp2_b.png"**

Note : This code may take upto 15 secs to return the results , since it is executing StrategyLearner code 8 times for different impact values.

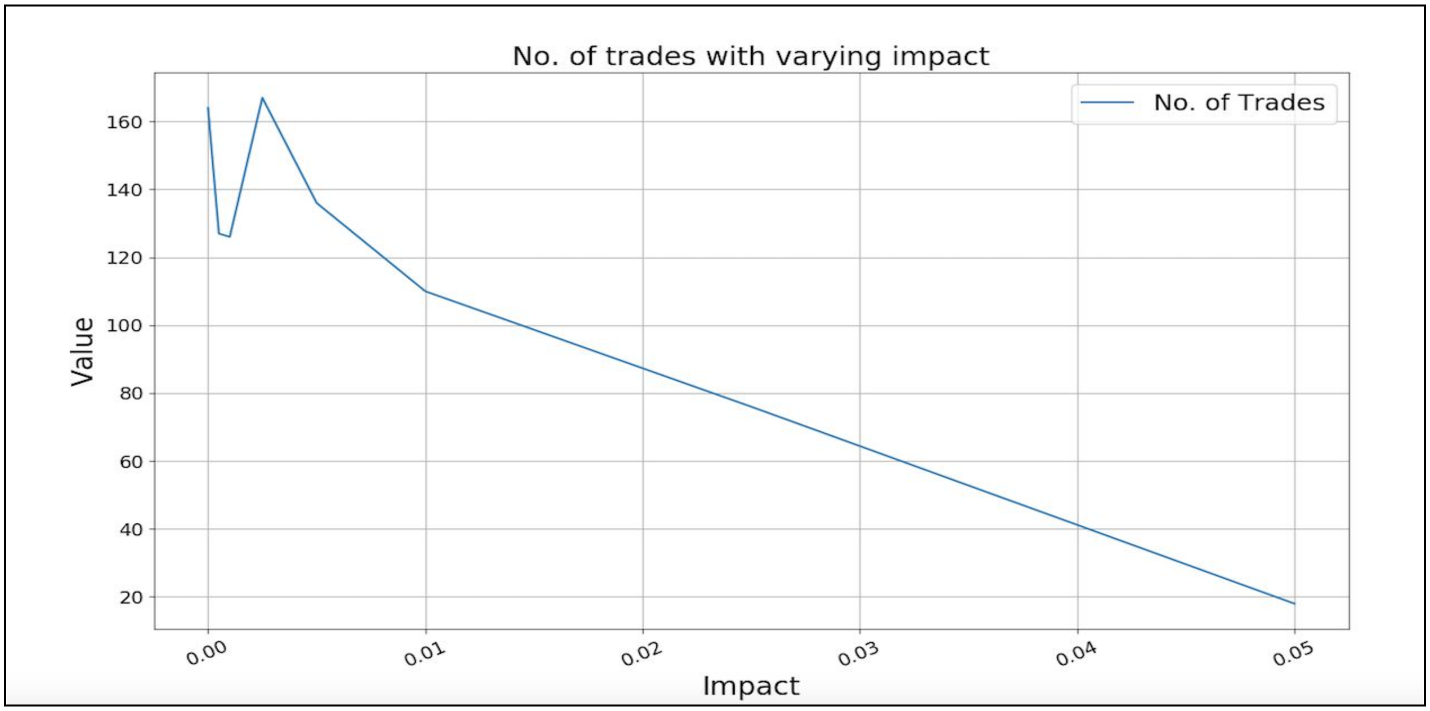
Analysis :

Below graph shows varying values of cumulative return and no. of trades with varying values of impact.

Experiment2.py generated graph : Exp2_a.png



Experiment2.py generated graph : Exp2_b.png



As the definition of market impact goes:

Market Impact is the extent to which the buying or selling moves the price against the buyer or seller, i.e., upward when buying and downward when selling. It is closely related to market liquidity; in many cases "liquidity" and "market impact" are synonymous.

We can clearly see that No. of Trades reduces significantly for the large increases in Market Impact. This can be justified that with increasing value of market impact , people tend to trade less frequently , as there is more cost involved for the transaction , making the market less liquid.

We can also see from the above graphs that i**f there are significant increase in market impact values ,the "Cumulative Return" of the stock shows decreasing trend**. However **very small changes in impact cost , does not necessarily affect the returns** .

So high impact cost does not only reduces the returns per trade , but also frequency of trading which in turn results in lower cumulative returns.