Project 8 : Strategy Evaluation (CS7646)

Stella Lai-Hoong Soh

lsoh3@gatech.edu

An Overview of Indicators

The indicators I have chosen to use are: Simple Moving Average(SMA), Exponential Moving Average(EMA) and Bollinger Band Percentage (BBP).

Simple Moving Average (SMA)

A Simple Moving Average (SMA) is the average price of a stock symbol over a lookback period or a selected range of time. It is calculated by totaling up all the closing prices of a security over a period of time and dividing by that number of time periods.

Exponential Moving Average (EMA)

Exponential Moving Average (EMA) places greater weight and significance on recent prices, as opposed to Simple Moving Average (SMA) which relies heavily on historical data. EMA's change faster than SMA's, and could serve as more timely indicators of the market.

Bollinger Band Percentage (BBP)

Bollinger Band Percentage (BBP) quantifies the relationship between price and Bollinger bands, and is defined as follows:

BBP (or %B) = (Price - lower Bollinger band) / (Upper Bollinger band - Lower Bollinger band)

As shown in my Manual Strategy code, I have used BBP to identify overbought and oversold situations.

1. Manual Strategy

First we read in the prices of the symbol under test for the specified time period of observation. After filling forward (‘ffill’) and backward(‘bfill’) to work around the gaps, I derived df_prices, the Dataframe to be used. I have chosen the lookback window to be 10 days. The SMA, EMA, and BBP values were computed by calling on indicators.py’s cal_SMA(), cal_exp_ma() and cal_Bollinger_bands() respectively on df_prices for that lookback window. Our desired_position is then a copy of df_prices. I iterated over the rows of desired_position as (date, data) pairs. My code examines the data in the following situations:

(A) If any of the SMA, EMA or BBP values at a particular date shows NaN value, a 0 is designated to desired_position at that date.

(B) If df_prices is higher than Bollinger upper band (i.e. a Bollinger band crossing), this is an overbought situation. The security is trading at a level above its intrinsic or fair value. This stock may be a good candidate for sale. We indicate in the code by designating the desired_position value at this date to be -1000.

(C) If df_prices is lower than Bollinger lower band (i.e. a Bollinger band crossing), this is an oversold position. The security is worth more than its current fair value. This is a BUY signal for the trader. We indicate in the code by designating that desired_position at that date to be +1000.

(D) The EMA line pushes past the SMA line, and we see an uptrend of the stock. This is a BUY signal. We indicate in the code by designating that desired_position at that date to be +1000.

(E) The EMA line falls below the SMA line, and we see a downtrend of the stock. This is a SELL signal. We indicate in the code by designating that desired_position at that date to be -1000.

Now that the desired_position Dataframe is built up, the trades Dataframe is obtained by taking the difference (i.e. doing a diff) between elements of desired_position Dataframe.

With the trades Dataframe in place, my test_code() function processes the in-sample trades for the period of January 1, 2008 to December 31, 2009, and the out-of-sample

trades for the period of January 1, 2010 to December 31, 2011. The respective portfolio statistics and charts are plotted for each period.

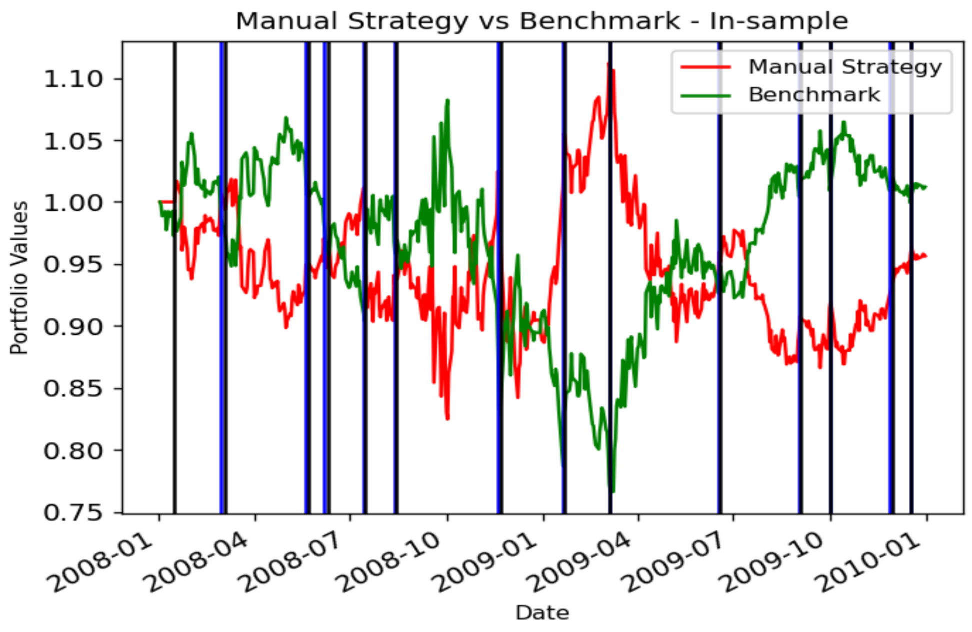


Figure 1.1 — Chart of portfolio values of Manual Strategy vs. Benchmark for in-sample period

Figure 1.1 above shows the chart of normalized portfolio values of Manual Strategy versus Benchmark for the in-sample period (1/1/2008 to 12/31/2009). The vertical blue lines indicate the LONG (i.e. BUY) entry points and black lines indicate SHORT(i.e. SELL) entry points.

As shown in Figure 1.1, there were periods of time (from the beginning of 2008 until around the end of 2008) whereby the Benchmark outperforms the Manual Strategy trading. From January 2009 till around August 2009, the Manual Strategy, however, outperforms the Benchmark.

Tables 1 and 2 below confirm this observation as well: **In-sample**

Portfolio Statistics	**Manual Strategy**	**Benchmark**
Cumulative Returns	-0.0437	0.0123
Average Daily Returns	0.0001	0.0002
Standard Deviation of Daily Returns	0.0170	0.0170
Sharpe Ratio	0.0524	0.1569

Table 1— In-sample portfolio statistics for Manual Strategy vs.Benchmark

We can see above in Table 1- the in-sample case, the cumulative returns, average daily returns, standard deviation (similar values) and Sharpe ratio all indicate favorable returns relative to the risks the trader has taken if he/she adopts the Benchmark strategy (i.e. holding onto the stock).

Out-of-sample

Portfolio Statistics	**Manual Strategy**	**Benchmark**
Cumulative Returns	0.1011	-0.0834
Average Daily Returns	0.0002	-0.0001
Standard Deviation of Daily Returns	0.0078	0.0085
Sharpe Ratio	0.4531	-0.2568

Table 2— Out-of-sample portfolio statistics for Manual Strategy vs.Benchmark

In Table 2 - the out-of-sample case, the cumulative returns, the average daily returns, standard deviation and Sharpe ratio all indicate that the Manual Strategy is performing better than the Benchmark. Why does the Manual Strategy outperforms some of the time, but underperforms at other times?

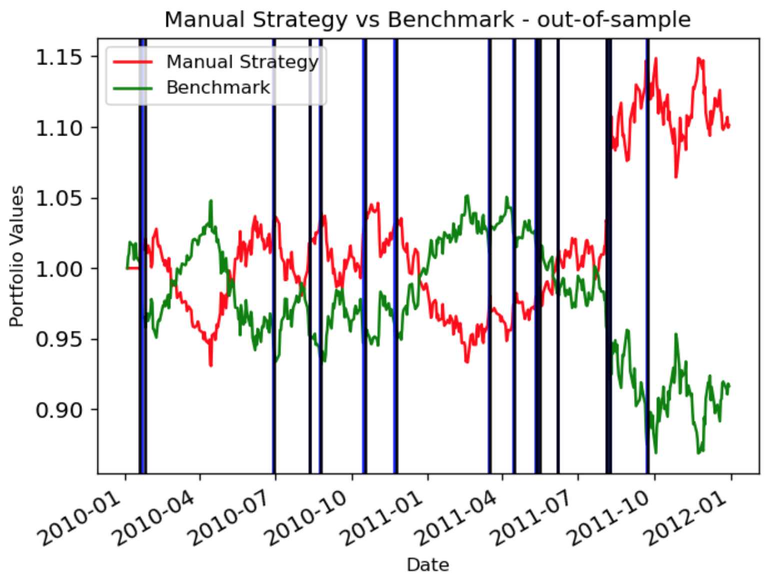


Figure 1.2 — Chart of portfolio values of Manual Strategy vs. Benchmark for out-of-sample period

The Manual Strategy calls for framing the problem as one of feeding features (in this case, our chosen indicators computed from df_prices), and making trading decisions based on Bollinger band crossings and EMA lines crossing the SMA lines. This makes for a rather crude trading strategy, that may sometimes work and sometimes may not. The philosophy of Benchmark entails holding onto a stock till the end of a trading period. When there are downtrends, this philosophy works well. However, it does not allow one to tap into gains should there be uptrends in the stock. In fact, we see this to be the case. A cursory glance into JPM.csv reveals that there was indeed a downtrend (i.e. prices going from the $40+ range to $30+ range and all the way to $20+ range) in stock prices from the beginning of 2008 until around the beginning of 2009. Figure 1.1 shows the Benchmark lines (in green) outperforming Manual Strategy. Table 1's portfolio statistics further confirm that during this period, the returns are much more favorable using Benchmark rather than Manual Strategy.

Given time and when there are upward swings in prices of the stock, as we head into July 2009 for example, the Benchmark trading style (i.e. holding onto a stock) could no longer hold its ground. Peeking into JPM.csv, I observed that indeed the prices of JPM went from a low $20+ range in April 2009 back to the $40+ range in July/August 2009. Hence, from the beginning of 2009 to July 2009, the Manual Strategy performs much better than the Benchmark trading style.

2. Strategy Learner

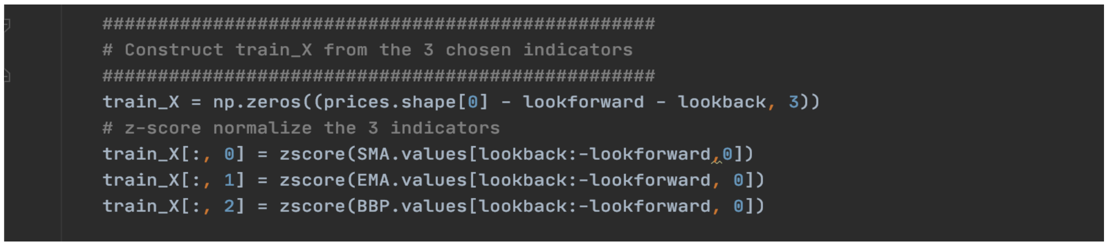
The Strategy Learner uses an available set of data as in-sample (or training) data to train the model. Once the model is trained, it then uses a new set of data, the out-of-sample (or testing) data to obtain a predicted Y value.

Phase 1

The constructor method in Strategy Learner creates an instance of a BagLearner with leaf_size of 5 and bags=20. The BagLearner goes on to create 20 instances of RTLearner.

Phase 2

In add_evidence() function, we define a lookback window of 10 days for computing SMA, EMA and BBP, and a lookforward window of 10 days for computing 10-day returns. After computing values of SMA, EMA and BBP, we construct train_X from the 3 chosen indicators. Thanks to vectorization, I was able to accomplish this in 4 lines of code:



```
##################################################
# Construct train_X from the 3 chosen indicators
##################################################
train_X = np.zeros((prices.shape[0] - lookforward - lookback, 3))
# z-score normalize the 3 indicators
train_X[:, 0] = zscore(SMA.values[lookback:-lookforward,0])
train_X[:, 1] = zscore(EMA.values[lookback:-lookforward, 0])
train_X[:, 2] = zscore(BBP.values[lookback:-lookforward, 0])
```

After train_X is constructed, my code classifies based on a 10-day change in prices. I first obtained the train_Y by computing future returns. YBUY has been set to impact value, and YSELL has been set to -YBUY. Once train_Y has been derived, train_Y values greater than YBUY are assigned a value of 1, values smaller than YSELL are assigned a value of -1, and the values between YBUY and YSELL are assigned 0.

Having now been equipped with train_X and train_Y, I then proceeded to train the learner.

Phase 3

In the testPolicy() function, we obtain a new set of prices for the out-of-sample period of January 1, 2010 to December 31, 2011. Once SMA, EMA and BBP have been computed based on this new set of prices, I constructed test_X. The query() method of the learner is then called upon to obtain test_Y.

The desired_position is 1000 * test_Y. I made a copy of desired_position and assigned to trades. The entire trades DataFrame is made up of np.diff(desired_position) assigned to the first entry onwards plus desired_position[0] written into the first entry.

Hyperparameters Tunings

Hyperparameters are points of flexibility that allow a model to better adapt itself to a given set of data. These model-specific properties are fixed before we start training or testing the model on a set of data. In Strategy Learner, I have used leaf_size of 5 and *lookback* and *lookforward* values of 10. *Lookforward* value is the "N"-days or number of days to look into the future.

Initially, I tried *lookback* and *lookforward* values of 20. That caused one testcase in grade_strategy_learner.py to fail. I then fine-tuned some more, and found that *lookback* and *lookforward* values of 10 work best as the model yielded the highest cumulative returns for that value.

Adjustment of Data
By constructing train_X and then train_Y and feeding them into 20 instances of RTLearner (or Random Tree Learner), we are using an ensemble technique of BagLearner with RTLearner. I did not have to discretize the data before the learner could be trained.

My code works by classifying based on "N"-day changes in price. When query() is called, the build_tree() function in RTLearner.py calculates the mode of the input data. Therefore, no adjustment of data was necessary for Strategy Learner to function.

3. Experiment 1

Experiment 1 compares the performance of **Manual Strategy**, **Strategy Learner** and **Benchmark** (i.e. buying 1000 shares on the first trading day and holding onto till the last trading day). Strategy Learner has an in-sample (training) period of January 1, 2008 to December 31, 2009 and an out-of-sample (testing) period of January 1, 2010 to December 31, 2011. I trained the **Strategy Learner** in the in-sample period using add_evidence() and tested it in the out-of-sample period using testPolicy(). The resulting sl_trades DataFrame obtained is then fed into marketsimcode.py's compute_portvals().

With **Manual Strategy,** I used the out-of-sample period (so that we have the same comparison base) to compute ms_trades. The resulting ms_trades DataFrame is then fed into marksetsimcode.py's compute_portvals() and the portfolio values of Manual Strategy for this period is obtained.

My benchmark_trades is simply a copy of ms_trades. After setting the condition of buying 1000 shares of the benchmark stock and holding till the last trading day, I computed benchmark_portvals, the portfolio values of **Benchmark** by invoking marketsimcode.py's compute_portvals().

Portfolio Statistics	**Strategy Learner**	**Manual Strategy**	**Benchmark**
Cumulative Returns	0.1343	0.1011	0.0123
Average Daily Returns	0.0003	0.0002	0.0002

Standard Deviation of Daily Returns	0.0073	0.0078	0.0170
Sharpe Ratio	0.6051	0.4531	0.1569

Table 3—Portfolio statistics for Strategy Learner vs. Manual Strategy vs.Benchmark

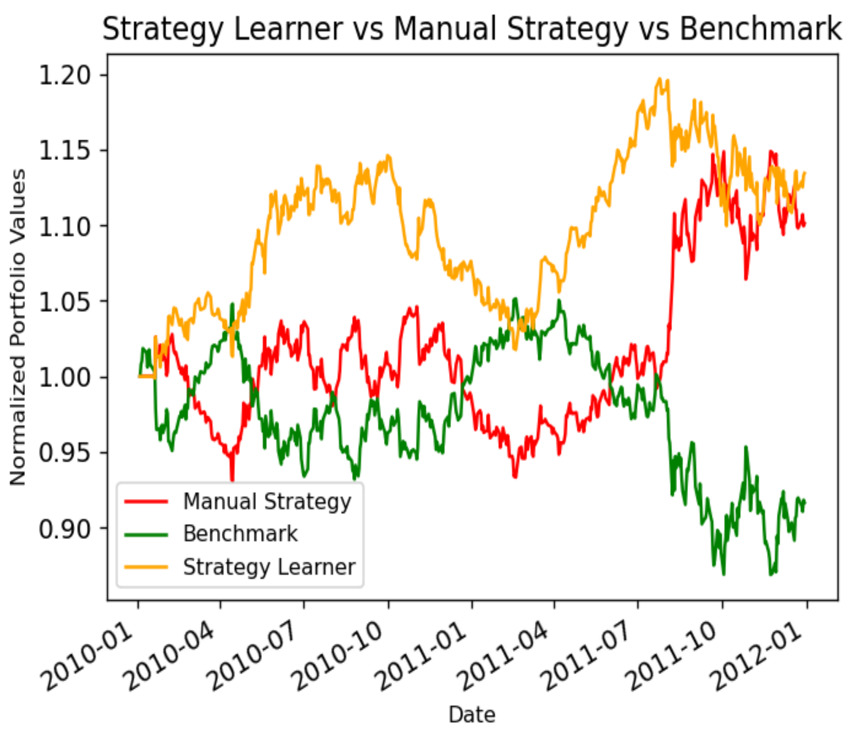


Figure 3 — Chart of portfolio values of Strategy Learner vs. Manual Strategy vs. Benchmark

From Figure 3 above, we see that Strategy Learner outperforms both the **Manual Strategy** and **Benchmark** for the out-of-sample period. The high Sharpe ratio of 0.6051 indicates that amongst the 3 strategies, the **Strategy Learner**'s returns are much more favorable relative to the risks it has taken.

Summary of the Outcome

My expectation was the Strategy Learner would perform better than the Manual Strategy and Benchmark. Looking at Figure 3 above, the Strategy Learner indeed outperforms both Manual Strategy and Benchmark. In Manual Strategy, trading decisions are made based on 3 indicator values - SMA, EMA and BBP. The Strategy Learner, on the other hand, uses an ensemble technique, which along with indicators, has fewer errors and offers less overfitting. This in turn means Strategy Learner gives better performance on a given set of data.

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

The Strategy Learner uses 20 “bags” or instances of Random Tree learners in the ensemble technique. While there is sophistication in terms of training a given set of data and then testing with a new set of data, the randomness that is introduced could mean there is a chance for less-than-optimal results sometimes. Therefore I would not expect this relative result every time with in-sample data.

4. Experiment 2

Market impact is the extent to which a price moves against the trader whenever he/she buys or sells a stock. It moves upwards when buying occurs and downwards for a sale. Essentially, this is a cost the trader incurs due to market slippage or plainly, the transaction itself changes the price of the asset. Therefore, my hypothesis is: the lower the impact value, the more favorable it is a stock transaction for a trader. This translates to greater values of **cumulative returns**, **average daily returns** and **final portfolio value**.

I have designed experiment 2 to compare the normalized portfolio values of Strategy Learner for 3 impact values: 0.0025, 0.005 and 0.05 in the in-sample trading period of January 1, 2008 to December 12, 2009. The 3 Strategy Learners with different impact values are trained using add_evidence() method. Once the learners are trained, they are tested using testPolicy(), which calls on RTLearner’s query() to obtain test_Y value. Allowable positions could be 1000 shares long (BUY), 0 shares (HOLD) or -1000 shares short (SELL). Correspondingly, trades could be between buying 2000 shares or selling 2000 shares.

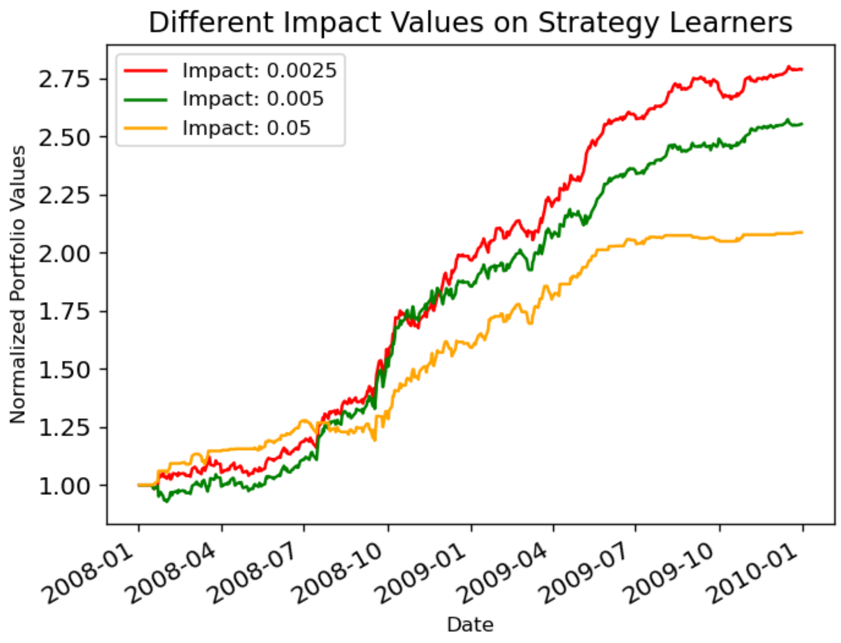


Figure 4 — Chart of different impact values on Strategy Learners

Figure 4 above indeed shows that the impact value of 0.0025 (in red) yields a greater return on portfolio value.

Portfolio Statistics	**Impact = 0.0025**	**Impact=0.005**	**Impact=0.05**
Cumulative Returns	1.7879	1.5527	1.0865
Average Daily Returns	0.0021	0.0019	0.0015
Standard Deviation of Daily Returns	0.0100	0.0105	0.0089
Final Portfolio Value	278790.0	255270.0	208650.0

Table 4 —Portfolio statistics for different impact values on Strategy Learners

Table 4 above confirms that amongst the 3 impact values of 0.0025, 0.005 and 0.05, impact value of 0.0025 yields the greatest cumulative returns, average daily returns and final portfolio value.