ML4T - Assignment 8 - Strategy Learner

Shichao (Michael) Liang
GTID: 903540912

December 1, 2019

1 Introduction

For this project, we design a learning trading agent using a classification-based learner. In particular, we'll generate a strategy using a bootstrapped ensemble of Random Forest learners. We will then determine in-sample performance versus a buy-and-hold benchmark, as well as investigate the effect of impact cost on performance metrics.

2 Framing the Trading Problem

In order to use our ensemble of Random Forest learners to decide on a trading strategy, we will have to reformulate the trading problem into a learning problem. In particular, we have to reformulate our technical indicators and fundamental analysis into training features and classes, and once queried output a list of trades:

1. **Features** - For our problem, we will specify technical indicators as features on which our classification learner will train. These are detailed in Section 2.1.

2. **Classes** - For the corresponding classes, we will assume that for the stock we're learning, we can be in one of three positions: **LONG**, **SHORT**, or **CASH**. In particular, given a max holdings of 1000 shares of the stock, LONG corresponds to holding the maximum possible amount of shares, SHORT corresponds to shorting the maximum possible amount of shares, and CASH corresponds to cashing out positions.

Using the above framework, we detail the steps to convert a trading problem into a learning problem in Section 2.2

2.1 Technical Indicators

For our features, we will choose to use three technical indicators:

1. **Simple Moving Average** - The mean of recent prices over a set time window n. It is commonly used to determine if an asset price will continue to follow a trend. Depending on the length of the time window, SMAs can smooth out volatility at the cost of added lag between the SMA data and the source data. We will discretize/standardize our data by utilizing the price-to-SMA ratio at time step t, which is described below:

$$\mu_{SM}(t) = \frac{p_t \times n}{p_t + \cdots + p_{t-(n-1)}}$$

This new indicator provides better trading signals. With our manual trading strategy, any ratio above 1 will indicate that the price is above the moving average, and thus indicates a signal to sell. Likewise, a signal below 1 will indicate that the price is below the average, and thus indicates a signal to buy. For our learner, we will utilize a time window $n = 10$ where a time period is 1 day.

2. **Momentum** - The rate at which prices change over a period of time. Since momentum is a rate of change, it's important to specify a time window n. We will discretize/standardize our indicator data by calculating the momentum's rate of change:

$$\delta_{momentum}(t) = \frac{price(t)}{price(t-n)} - 1$$

This value is centered around 0, with higher values indicating an bullish trend, and lower values indicating a bearish trend.

3. **Bollinger Bands®** - A type of indicator that characterize the prices and volatility over time of an asset using an envelope of the maximum and minimum of the moving average. In general, the upper band and the lower band are k standard deviations away from the moving average (typically $k = 2$). We will discretize/standardize our data by calculating a price's deviaton from the SMA as a fraction of the Bollinger Bands®:

$$BB_{ratio}(t) = \frac{price(t) - \mu_{SM}(t)}{k \times \sigma(t)}$$

Thus, a ratio of 1 indicates that the price has reached the upper bound of the Bollinger Band®, or that the asset has been overbought. Likewise, a ratio of -1 indicates that the price has reached the lower bound of the Bollinger Band®, and indicates that the asset has been oversold.

2.2 Steps for the Learning Problem

In order to convert a trading into a learning problem, we created a StrategyLearner, which converts the problem in three steps:

1. **Setting up the Learner Parameters** - In this step, we initialize an instance of our BagLearner using the following hyperparameters:
   (a) leaf-size - 5
   (b) bag-size - 20
   (c) learner - Random Tree Learner (RTLearner)

2. **Training the Learner** - The StrategyLearner trains on a set of training data. Since we are employing bootstrap aggregation, 20 separate RTLearners are trained on data randomly-selected from the training data set with replacement.
   (a) *Generate Training Features* - Our training data is an $n \times 3$ DataFrame of values corresponding to the indicators described in Section 2.1, where n describes the number of trading days.

(b) *Generate Training Classes* - The 1-day percentage change in price is used as daily returns, which are transformed into classes for our training data set. Positive percentage changes that are profitable (above impact cost) translate into LONG positions, negative percentage changes that are profitable (below negative impact cost) translate into SHORT poitions, and all other days are CASH positions.

(c) *Train the Ensemble* - Use the features and classes to train the ensemble of 20 RTLearners.

3. **Querying on Testing Data** - When provided with a testing data date-range, the StrategyLearner will return a list of trades by querying the ensemble of RTLearners, using the mode of the ensemble results.

   (a) *Generate Testing Features* - Our testing features set is an $n \times 3$ DataFrame of values corresponding to the indicators described in Section 2.1, where n describes the number of trading days.

   (b) *Generate Testing Classes* - Classify each trading day in the testing date-range by querying the ensemble of learners to get the mode of class results. Positive results are converted to LONG positions, negative results are converted to SHORT positions, and all else are converted to CASH positions.

   (c) *Generate Trades* - From the DataFrame of positions, generate a DataFrame of trades that will result in the desired position. Since the only allowed holdings for positions are in the set $\{-1000, 0, 1000\}$, trades must be within the set $\{-2000, -1000, 0, 1000, 2000\}$. Output this set of trades.

3 Experiment 1

In Experiment 1, we wish to investigate the performance metrics of our StrategyLearner on in-sample data. We operate under the following assumptions:

- **Symbol**: "JPM"
- **Starting Date**: 1 January 2008
- **Ending Date**: 31 December 2009
- **Commission**: \$0.00
- **Impact Cost**: 0.000
- **Starting Cash**: \$100,000
- **Allowable Positions in Shares**: $LONG = 1000$, $SHORT = -1000$, and $CASH = 0$
- **Ensemble Size**: 20
- **Leaf Size**: 5

Following the steps outline in Section 2.2, we train our StrategyLearner on the data from the given date range. Then, we query the same range to generate a list of trades, and calculate portfolio metrics.

We compare these metrics against those of a buy-and-hold benchmark, as well as against those of a manually-programmed strategy using the same indicators and thresholds. The portfolio values

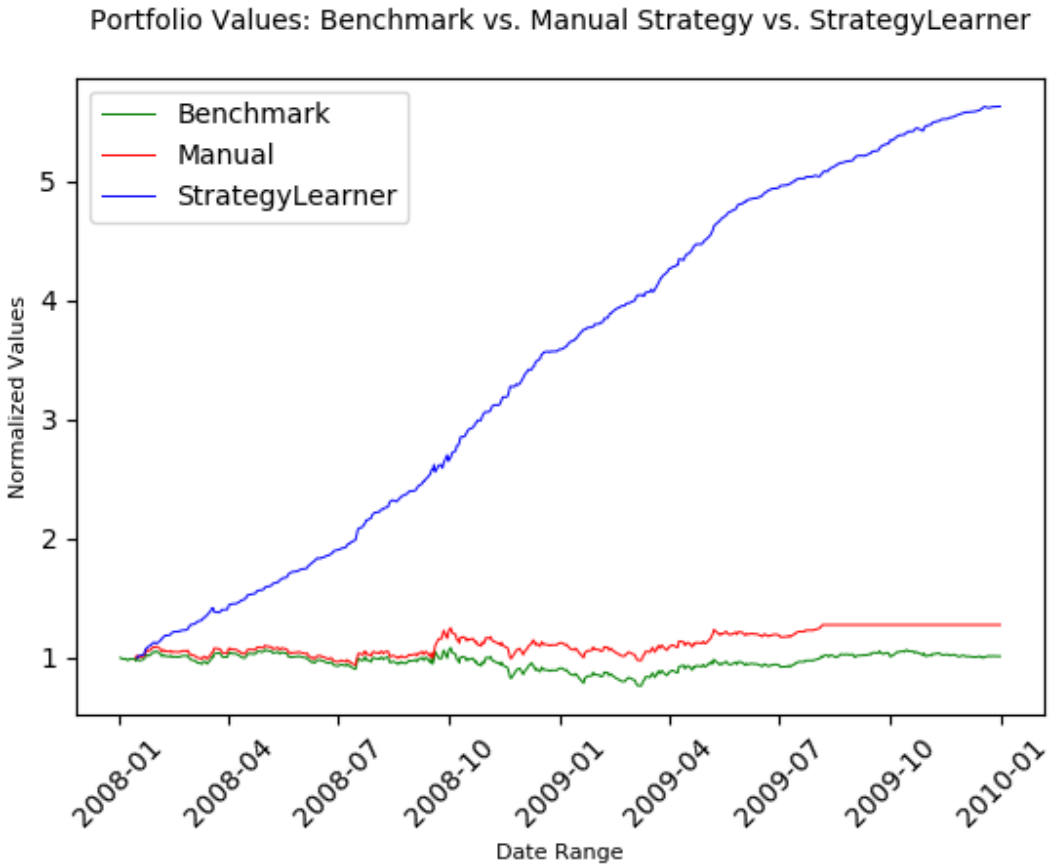

Figure 1: In-sample portfolio values over time for StrategyLearner compared against a manual strategy and the benchmark.

	StrategyLearner	Manual Strategy	Benchmark
Cumulative Return	4.6297	0.2765	0.0123
$\mu_{\text{Daily Returns}}$	0.0035	0.0008	0.0002
$\sigma_{\text{Daily Returns}}$	0.0053	0.0163	0.0170
Sharpe Ratio	10.4682	0.7342	0.1569

Table 1: Performance criteria for StrategyLearner vs. Manual Strategy and Benchmark

over time can be seen in Figure 1. In addition, we show the quantitative return and Sharpe ratio metrics in Table 1.

We can see that the StrategyLearner is much more performant in the in-sample date range than both the manual strategy and the benchmark. The ensemble of learners is able to effectively learn the best features to split in order to classify indicator information. This should be expected for all in-sample data sets, as the StrategyLearner is well-fitted with respect to the in-sample data set using a strong classification-learner. However, since both the bootstrap aggregation technique and the Random Tree learner use randomization, there is the possibility that our learner will not be as performant. This is ameliorated by the large number of learners in the ensemble. In addition, out-of-sample data sets might show different results, but that is outside the scope of this experiment.

4 Experiment 2

In Experiment 2, we hypothesize that increasing the *impact cost* will decrease the cumulative return, the Sharpe ratio, and increase the standard deviation of daily returns on the in-sample data. Higher impact cost should result in lower cumulative return due to each transaction incurring a financial cost. This will in turn lead to a lower Sharpe ratio. In addition, higher costs associated with transactions increases the volatility of the daily returns, which means we should see an increase in the standard deviation.

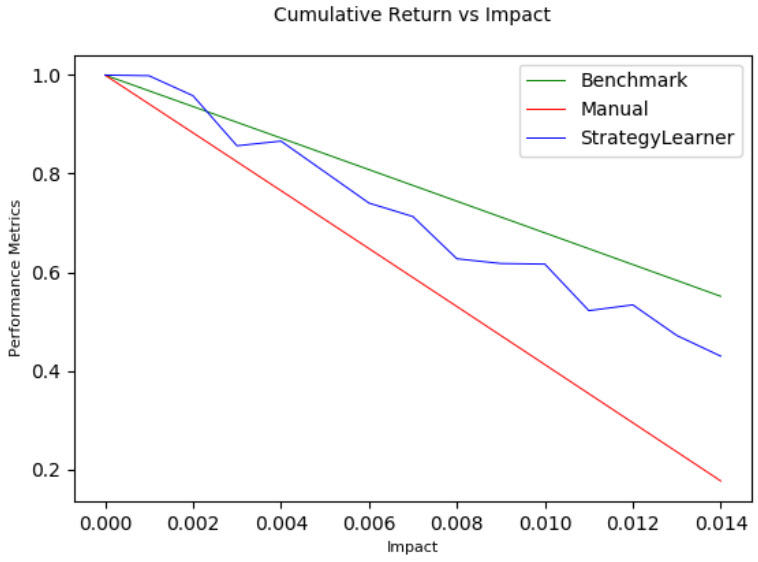

Figure 2: Cumulative return generally decreases as impact cost increases.

To conduct the experiment, the StrategyLearner was trained using the same parameters described in Section 3, except the impact cost was varied in increments of 0.001 from $[0.000, 0.015)$. Then, performance metrics were calculated using these impact costs for the StrategyLearner, the manual strategy, and the benchmark.

For the three metrics investigated, the results were normalized against the metric result with impact cost of 0.000. These are plotted in Figures 2, 3, and 4. The results do indeed confirm the hypothesis, not only for the StrategyLearner, but also for the manual strategy and the benchmark.

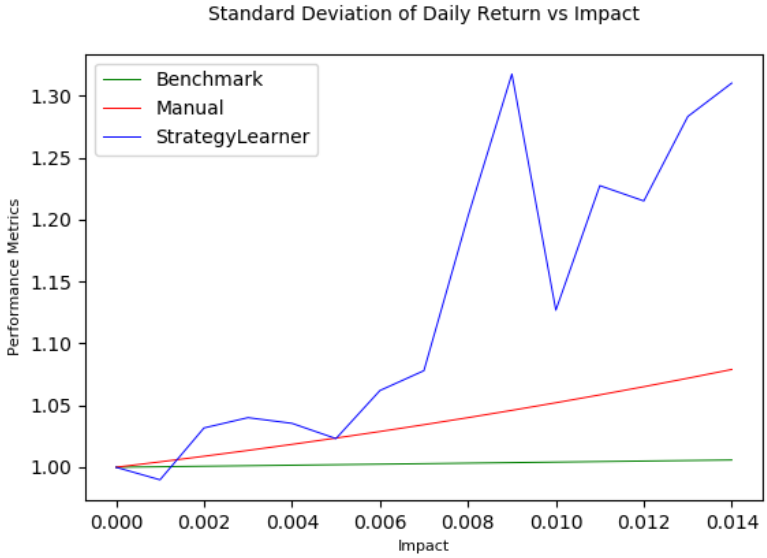

Figure 3: Standard deviation of the daily return generally increaases as impact cost increases.

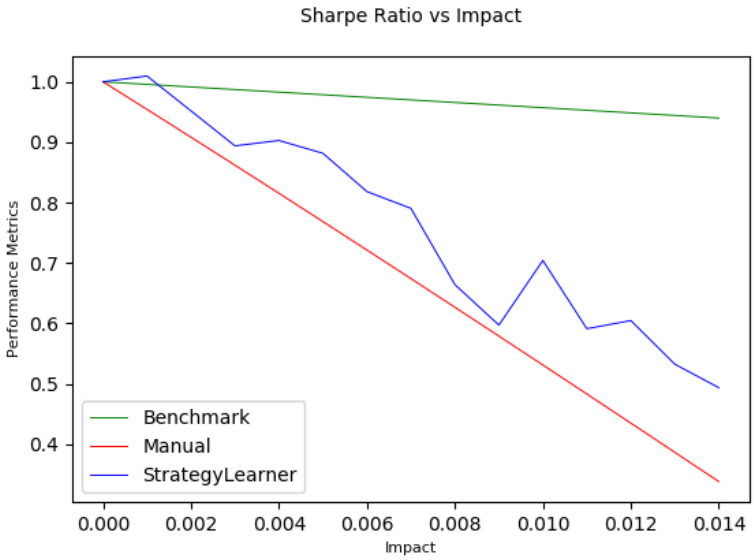

Figure 4: Sharpe ratio generally decreases as impact cost increases, which is directly tied to cumulative return.