Reinforcement Learning Trading System

By Jay Teguh Wijaya Purwanto (teguhwpurwanto@gmail.com)

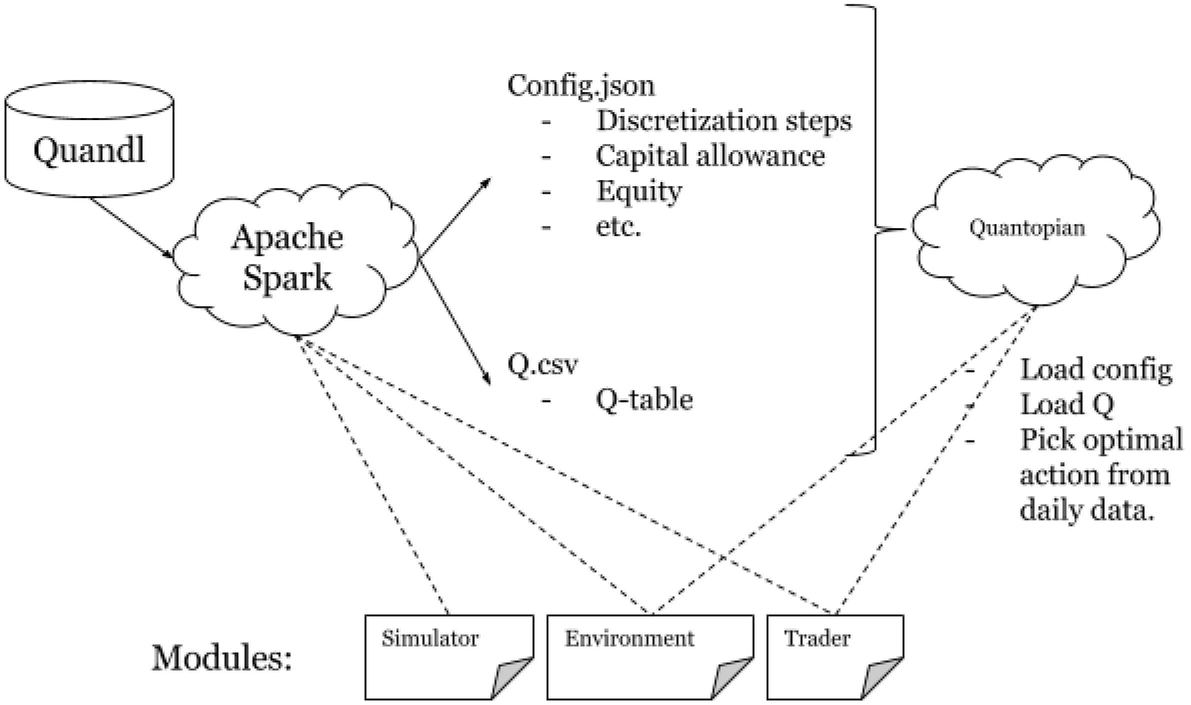
Abstract

This project aims to create a simple trading system with reinforcement learning, specifically Q-learning algorithm.

Definition

Project Overview

The trader monitors only a single equity. We use TSLA stock from date 2014-05-09 to 2015-02-27 for training and from 2015-03-03 to 2015-03-27 for validation testing (. Quandl's free stock data service was used to provide the data in this project. Apache Spark for Python was used to write the code to facilitate several traders (the term *trader* and *agent* are used interchangeably throughout this report) working in parallel each monitoring a different equity.



The diagram above shows this system in more detail. Our Apache Spark server loads the data from Quandl server, which will then use modules Environment, Trader, and Simulator to produce a config file and a Q-table. They will then be passed on to Quantopian server in which similar Environment and Trader classes would load and use the files to pick an optimal action from daily data.

Problem Statement

This project builds a reinforcement learning system that uses past six months of data to pick the optimal actions for the next 20 trading days.

The author tested three slightly differing scenarios as follow:

1. **Scenario 1:** As a baseline, the agent picks the actions at random. We want to see if the learning trader would perform better than a random walk.
2. **Scenario 2:** Stops learning after the training period ends i.e. by setting its learning, exploration, and discount rate to 0.
3. **Scenario 3:** Let the agent learn during testing phase by setting its learning, exploration, and discount rates to a small number.

Metrics

Various metrics were used to see which scenario had performed "better," ordered by their importance:

1. Alpha

Alpha metric measures performance of an investment on a risk-adjusted basis. The farther from 0 this value is, the more likely the result of the investment was not caused by market's movement and rather from the trader's skills. Value range is $-\infty$ to ∞, the higher the better.

Alpha is calculated by first fitting a linear regression model to daily returns of benchmark index fund (SPY in this case) against the portfolio, resulting in the following function:

$$y = \beta x + \alpha$$

The model's intercept is then the Alpha factor. The author argues that this is the most important metric in this project since it is more closely related to the trader's skills than other metrics.

2. Sharpe Ratio

This metric measures risk-adjusted return of a portfolio by subtracting daily risk-free rate from the daily return, find the mean of the results, then divide it by standard deviation of portfolio's returns. The higher an investment's Sharpe ratio is, the better it performed.

Sharpe Ratio was originally calculated annually, but in this project, we use a daily return instead of annual return. Due to this, the resulting value from the above needs to be multiplied by an adjustment factor of $\sqrt{\#\ samples\ per\ year}$ or $\sqrt{252}$ since there are 252 trading days in a year. The final formula used for Sharpe ratio calculation in this project is then:

$$S = \sqrt{252} \cdot \frac{E[R_p - R_f]}{std[R_p]}$$

Where R_p is the return of portfolio and R_f is a risk-free rate. E notation asserts that the algorithm uses past data to *expect* more or less similar values for future investments.

3. Beta

Beta measures how strongly correlated the investment's returns with those of benchmark index fund. From the linear regression formula similarly used in Alpha metric above, Beta is the coefficient of x.

When Beta is 1.0, that means the returns of traded security behaves the same way with that of SPY (the benchmark index fund). Less than 1.0 means the security is less volatile compared to SPY (i.e. downtrend and uptrend of the market do not affect the security by much). Conversely, when Beta value is larger than 1.0, it means the security is more volatile than the market. It is usually preferable to pick investments with smaller volatility (i.e. Beta closer to 0.0).

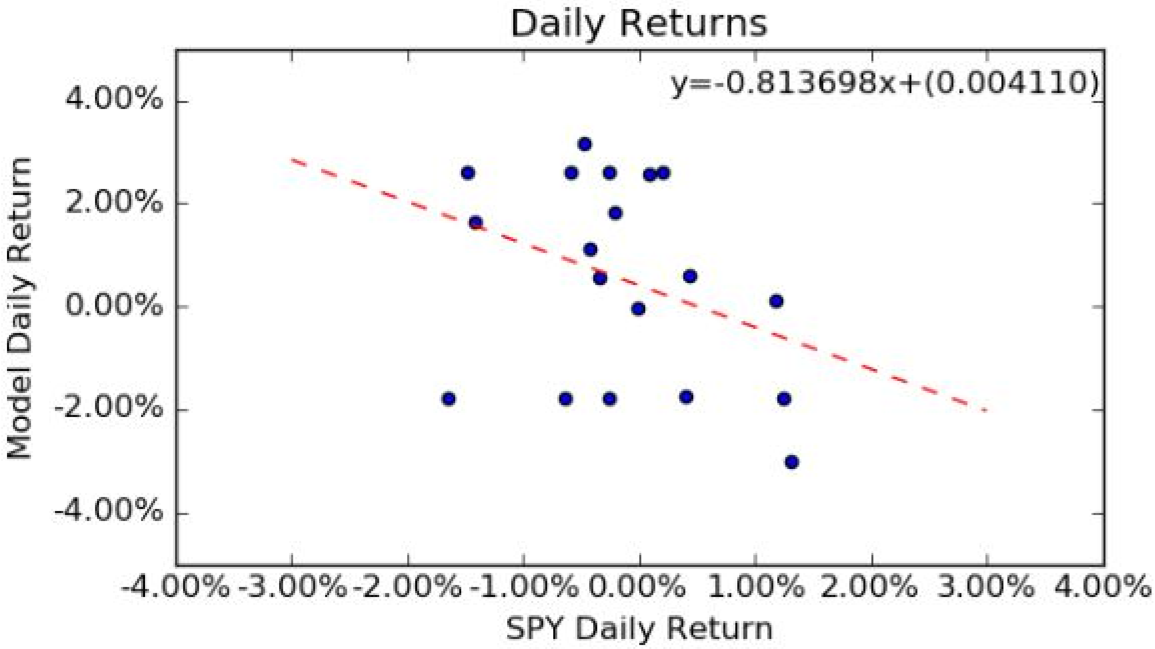


Fig 1. Fitting a linear regression model and using its coefficient and intercept as Beta and Alpha, respectively.

4. Total Returns

This metric measures the total returns accumulated by the trader throughout the testing period.

5. Max Dropdown

Max Dropdown is probably better measured by Alpha and Beta above, but this metric gives a quick intuition on how far did the investment value drop throughout the testing period.

Analysis

Data Exploration

The data used in this project are from TSLA stock from the period of 2014-05-09 to 2015-03-27. Validation testing uses data from the last 19 days (2015-03-03 to 2015-03-27), and the rest (2014-06-09 to 2015-02-27) for training, with the following details:

- There were 203 days in total.
- First 20 trading days were used in rolling window.
- The first day afterward was an input for the next day.
- In total, there were 183 days worth of data left used as training data, starting from 2014-06-09.

The following graph explains the reasoning for choosing this date range:

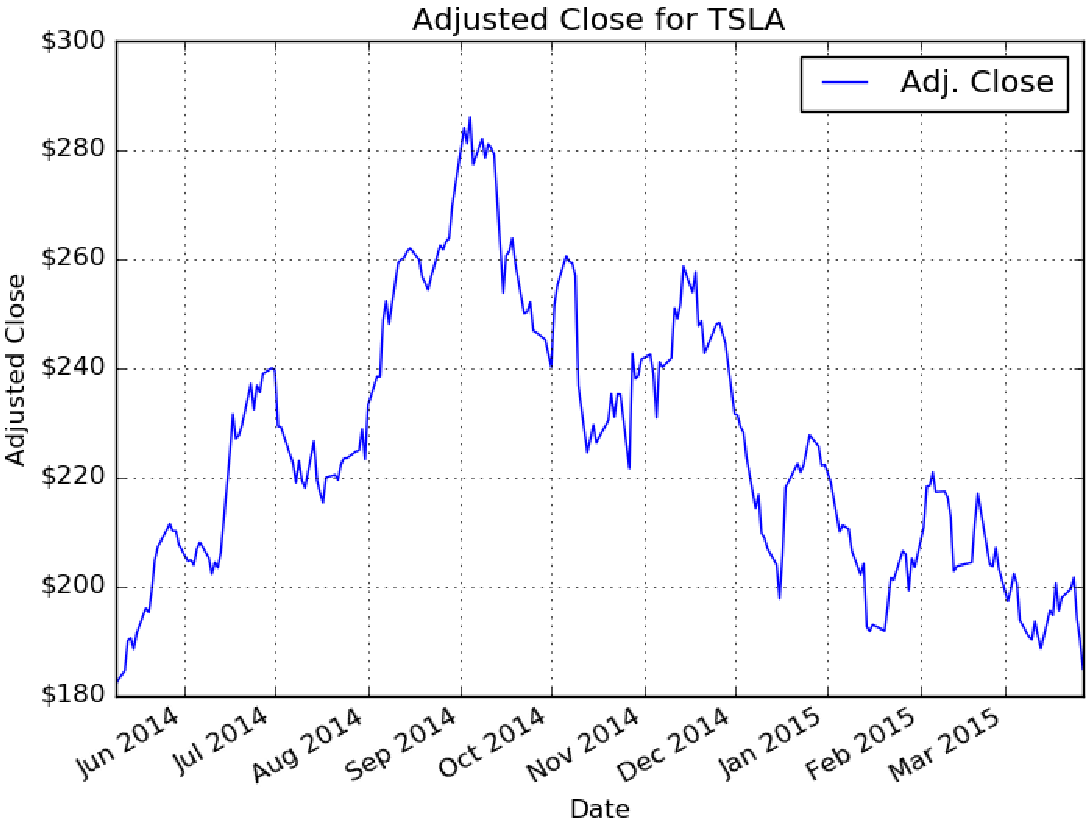


Fig 2. TSLA 09 Jun 2014 to 27 Mar 2015

Notice that the trend from May'14 to Dec'14 had a much higher magnitude than that of Jan'15 to Mar'15. In addition to that, Mar'15 did not appear to be a good time to trade as the overall trend was going downward.

The author was interested to see whether reinforcement learning could allow the agent to learn even from this seemingly suboptimal scenario. The following statistics further emphasize the differences between training and testing sets:

	Training (2014-06-09 to 2015-02-27)	**Testing** (2015-03-03 to 2015-03-27)
Num. days	183	19
Avg. close (adj.)	$231.08	$195.76
Max. close (adj.)	$286.04	$202.43
Min. close (adj.)	$182.26	$188.68
Std. dev.	$23.59	$4.29

Exploratory Visualization

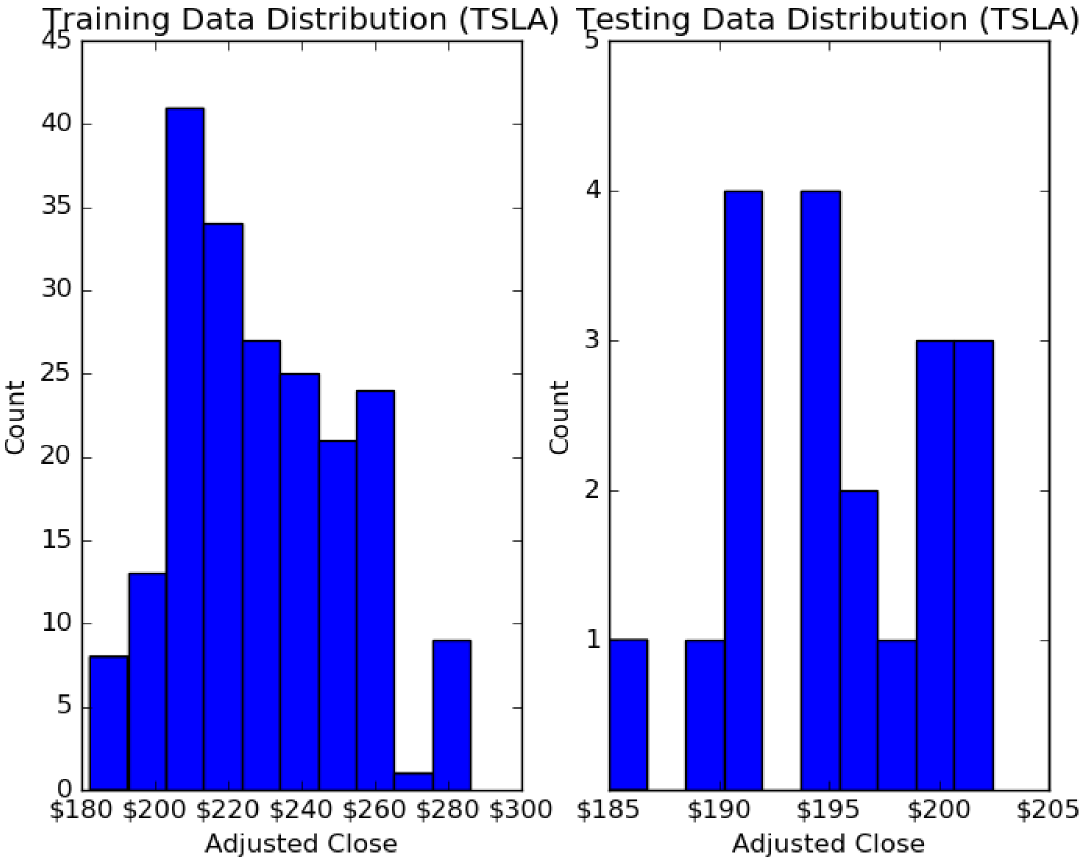


Fig 3. The distribution of training and testing data.

We see that for training data there are more values in the middle, and the training data covers all the values of the test data. This way the discretization step should have no problem in its implementation (will be explained in detail in later sections).

Let's also examine the daily returns of our data, compared to the returns of benchmark index fund SPY:

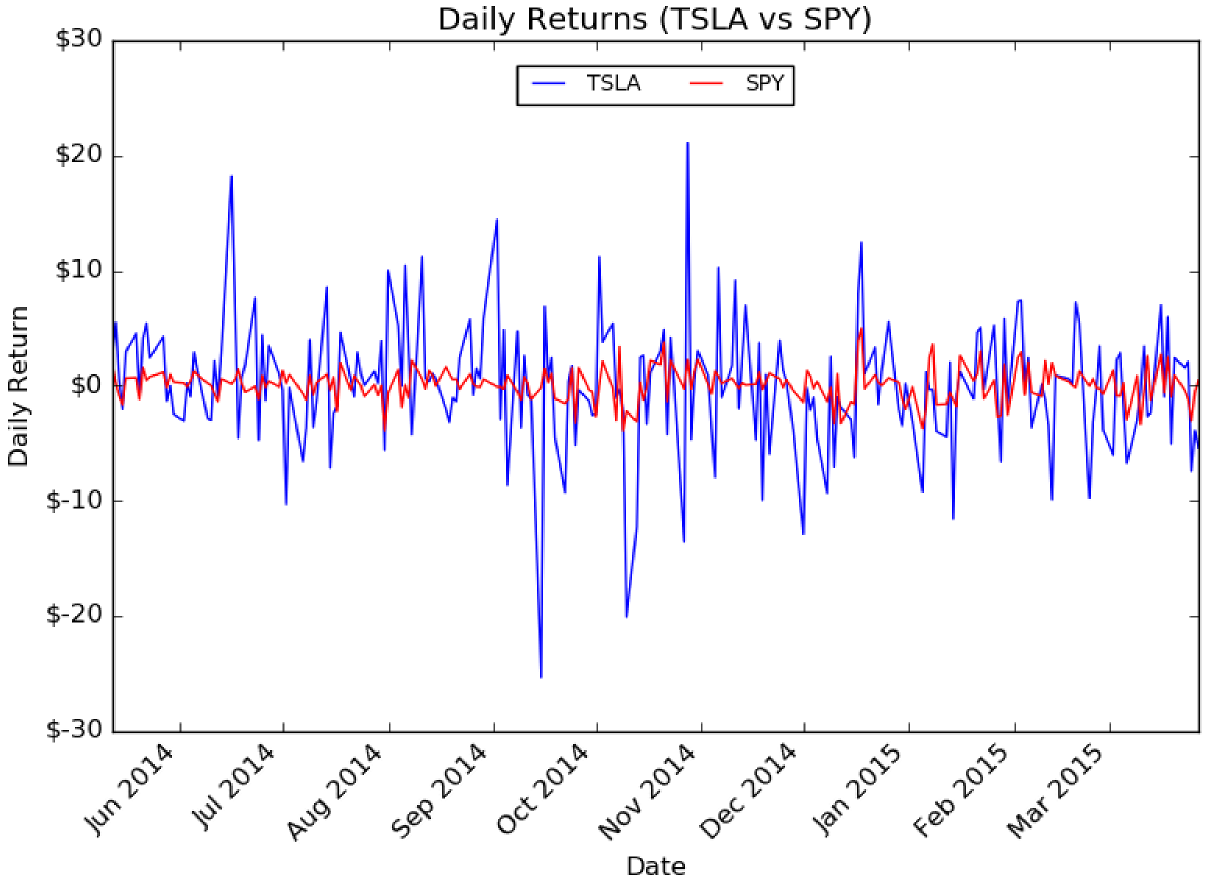


Fig 4. Daily returns of TSLA and SPY.

The daily returns, especially for TSLA stock, fluctuates wildly. To better capture the trend, we apply a 20-days rolling window. The resulting plot is as shown below.

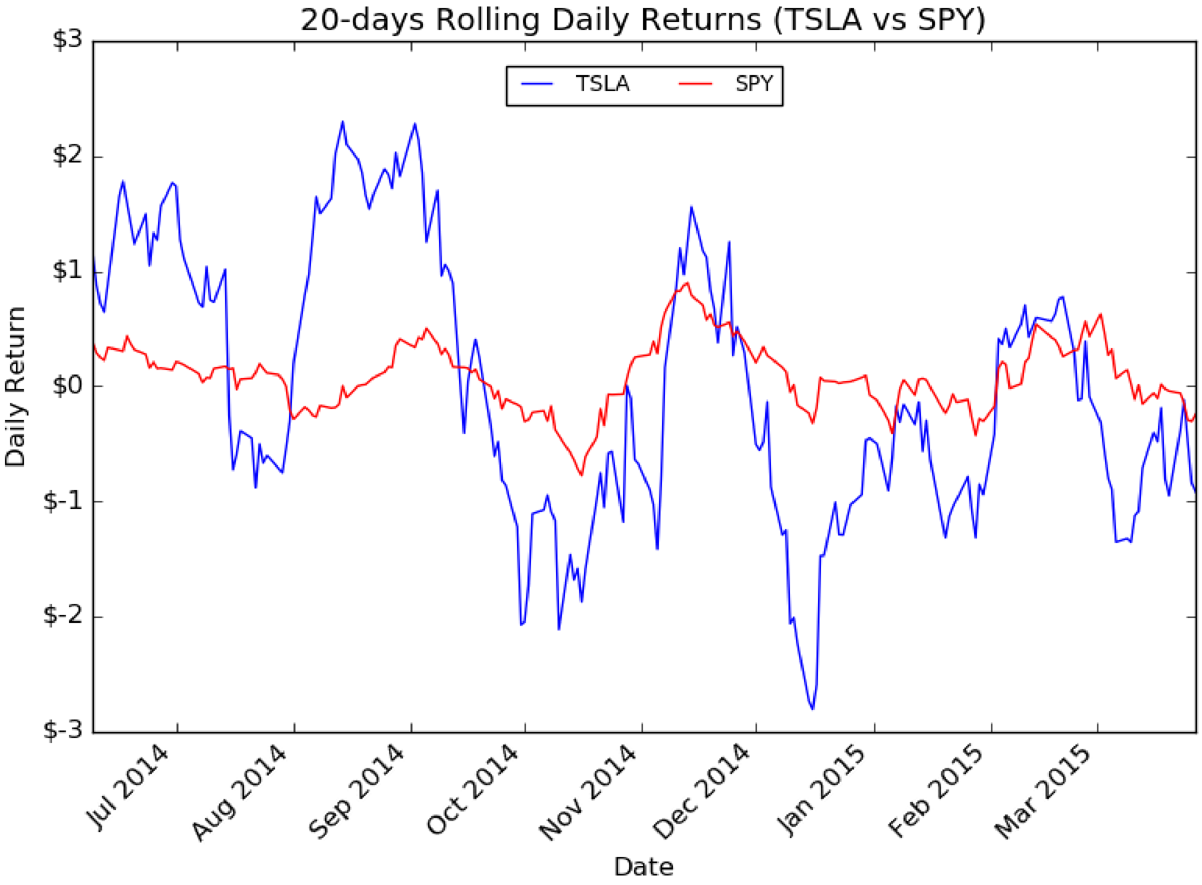


Fig 5. 20-days rolling daily returns of TSLA and SPY.

Notice that TSLA moves in similar directions, yet stronger, from SPY stock. Using the method described above, we can find out how strongly they correlate by plotting a scatterplot of daily returns and fitting a line regression model:

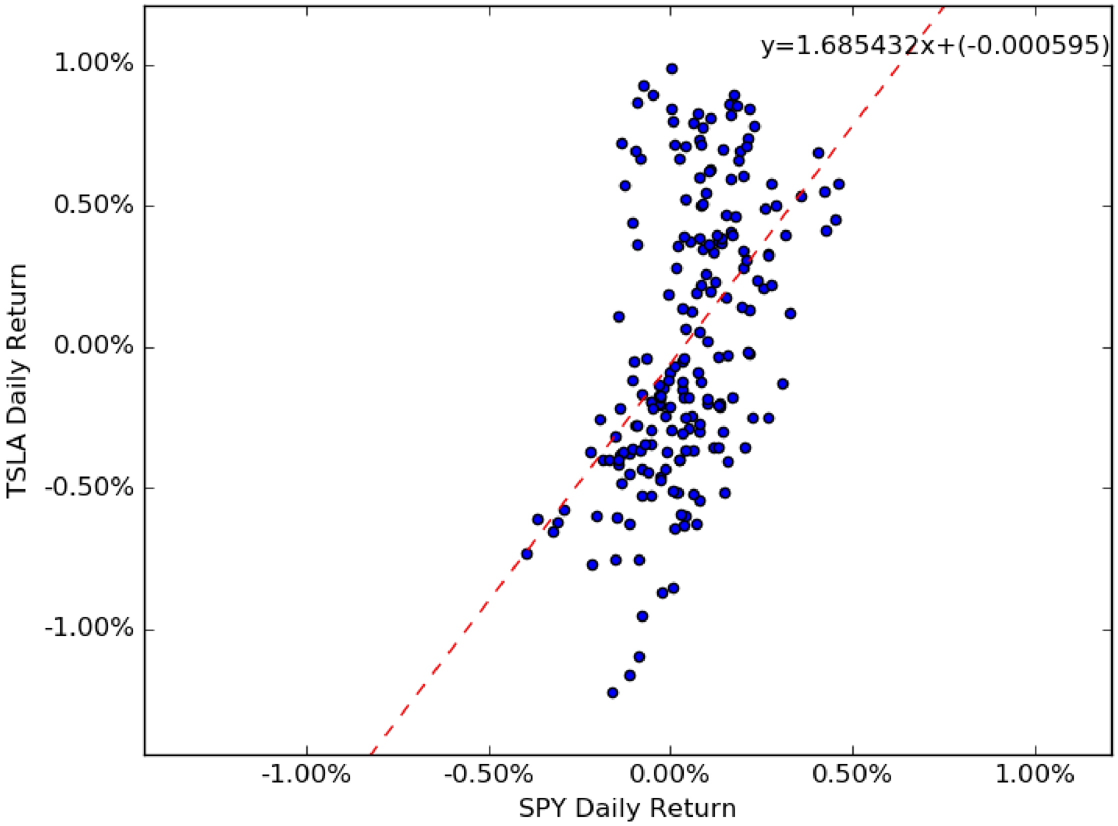


Fig 6. Correlation between daily returns of TSLA and SPY stocks.

Sure enough, the recorded high Beta value of 1.685 confirms the earlier observation.

Algorithms and Techniques

1. Q-learning

The learning agent uses the following Q-learning algorithm:

$$Q_{t+1}(s_t,a_t) \leftarrow \underbrace{Q_t(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha_t(s_t,a_t)}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{R_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\max_a Q_t(s_{t+1},a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q_t(s_t,a_t)}_{\text{old value}} \right)$$

Let's go through the variables one by one:

Q: This variable refers to the **Q-table**, a dictionary which stores the scores of actions the agent can take when faced with a given situation. This situation is called an **event**.

When given an **event**, the agent would pick an action that corresponds to the highest Q-value, or otherwise pick the action randomly when either of these two things happens:

1. Random occurrence based on ε (**Epsilon**) or **exploration rate** (explained below).
2. There are multiple actions sharing an identical maximum Q-value.

An **event** contains the following information:

- Adjusted Close Price
- Simple Moving Average (20-days rolling window)
- Adjusted Close Price / Simple Moving Average (20)
- Bollinger Bands (20)
- Daily Returns

Some of them are then used as the features to use in **state** creation by combining them together (explained in more detail below).

t: When learning, the agent is repeatedly going through the **events**, converting each of them to a **state**, and updating its **Q-table** on the cell that corresponds to that **state** and the chosen **action**. t refers to the previous **trial** while $t+1$ relates to the current **trial**.

s_t: A shorthand for a **state** at the (previous) **trial** t. To create this **state**, the author combined features from an **event** and other variables. The following list explains each feature used in a **state** and the justification of their inclusions:

1. Adjusted close and SMA ratio: This feature captures both the latest price and its trend.
2. Bollinger bands (upper and lower): These two features carry some information on whether the price was close to a turning point.
3. Daily return: This feature is relative against the previous date and *not the next date* since doing the latter would leak some future data into the trader's calculation.
4. Accumulated return since entry: This feature "teaches" the trader an inside perspective on when to sell currently held shares, an exact opposite to Bollinger bands.
5. Whether the trader is holding a share: This is a categorical feature that may help the agent to figure out when to buy and sell. Its values are:
    - Hold size > 0
    - Hold size == 0
    - Hold size < 0 (for short selling - not used for current project)

All of these features except number 5 consist of a potentially large range of values, so we would need to categorize them into smaller bins before combining them. This method is called *discretization,* and the next section explains this. An example value of a **state** is a series of integer, like *12210112101* (the next section below explains this).

a_t: An **action** at (previous) **trial** t. The value could be either *buy*, *hold*, or *sell*.

R_{t+1} : Reward of (current) **trial**. We calculate reward by multiplying *current* daily returns (not previous) by size of held stocks.

α : **Alpha** or **learning rate** is a rate that dictates how aggressive the agent should be when incorporating a new knowledge. The higher it is, the more likely the agent to pick new value compared to preserving old ones. Value is between 0 and 1.0.

γ : **Gamma** or **discount factor** is a factor with which we discount the knowledge from an estimated future value. Value is between 0 and 1.0.

ε : **Epsilon** or **exploration rate** is a parameter that ranges from 0 to 1.0 which dictates how likely the agent is to explore new actions. When the value is closer to 1.0, it means the agent would explore more. The formula above did not include this parameter.

One strategy to optimize the convergence of Q-table is by starting with a high value for **exploration rate**, and decay this value as the agent runs through more trials. This strategy effectively tells the agent to focus on exploration early on, and gradually change to exploiting the knowledge it has gathered over time. This strategy is called *Epsilon-greedy strategy*.

In this project, the author decided to use a cos adjusted epsilon decay (other decay algorithms were tested and reported in section "Refinement" below):

$$\epsilon = cos(0.05t) \text{ for } \varepsilon > 0.05$$

This function resulted in a convex-looking epsilon values, with higher values clustered in earlier trials. In the "Refinement" section, the author justifies the use of this function compared to other decay functions.

Final remark:
To conclude, the goal of training in Q-learning, in essence, is to make the **Q-table** converges. That means all, or at least most of the cells are filled with Q-values. We can then use this converged table to decide on which action to take when the agent experience an **event** by first mapping that **event** to a **state**, then use that **state** as a key in **Q-table** to pick an **action** with the highest Q-value.

Learning rate, **exploration rate**, and **discount factor** are parameters that we can adjust to change how the trader behaves.

2. Discretization

Discretization is an essential step in the state creation process. If we just use the values directly for the state creation, Q-table will not likely be usable for new data due to the huge variation of the states. Instead, we need to find a way to map an unseen value to a category that is "closer" to that value.

As a very simple example, imagine that we have the following values:
30, 31, 47, 211, 227, 228, 231, 600, 602, 619

If we were using three categories, the first category should contain [30, 31, 47], the second [211, 227, 228, 231], and the third one [600, 602, 619]. When we get a new value, of, say, 50, we can consider that new value belongs to category #1 due to its closeness.

Since there are multiple features per event, we would also then combine the calculated categories into a single integer number which is then used as the state to use in the Q-table.

In this project, the author had arbitrarily chosen to use 10 categories for each feature.

Benchmark

There are three essential characteristics to examine on each experiment to benchmark the performance of the model:

1. Compare Sharpe Ratio of the portfolio with that of SPY. If the former was higher, then we know that it is likely to be more profitable than SPY in the long term.
2. Positive alpha is good since it indicates that the gains were not (likely) due to the market's movement, but more due to the trader's performance.
3. -1.0 < Beta < 1.0 is good since that means the return of this portfolio is less volatile than that of SPY/the market.
4. Total Returns and Max. Drawdown are not necessary over the long term, but it is nice to know how the trader performed on the given test data.

Later sections present the reports of various benchmarks.

Methodology

Data Preprocessing

As explained in section "Algorithms and Techniques" above, the author uses the following features in this project:

- Adjusted Close Price / Simple Moving Average (20)
- Bollinger Bands (20)
- Daily Returns

This project downloads Daily Returns from Quandl server and process the other three from Adjusted Close Price. This section explains the creation of these three features.

1. Adjusted Close Price and SMA Ratio (20)

The author uses simple moving average instead of the adjusted close price so that the learning agent may learn from the underlying trend instead of the data directly. Simple Moving Average is a series of data calculated by creating series of averages of different subsets of the full data set.

SMA is calculated by first taking the first x number of a number series, average it, then "shift forward" i.e. excluding the first number and including the next number, and get the average of that second subset. We continue doing so until there is no more data point. The plot line connecting all the (fixed) averages is the moving average.

The model uses an arbitrarily chosen 20-day rolling window. Later, methods such as Akaike Information Criterion (AIC) or Bayesian Information Criterion (BIC) can be implemented to find the optimal number of rolling window ("Improvement" section below contains a deeper investigation into this).

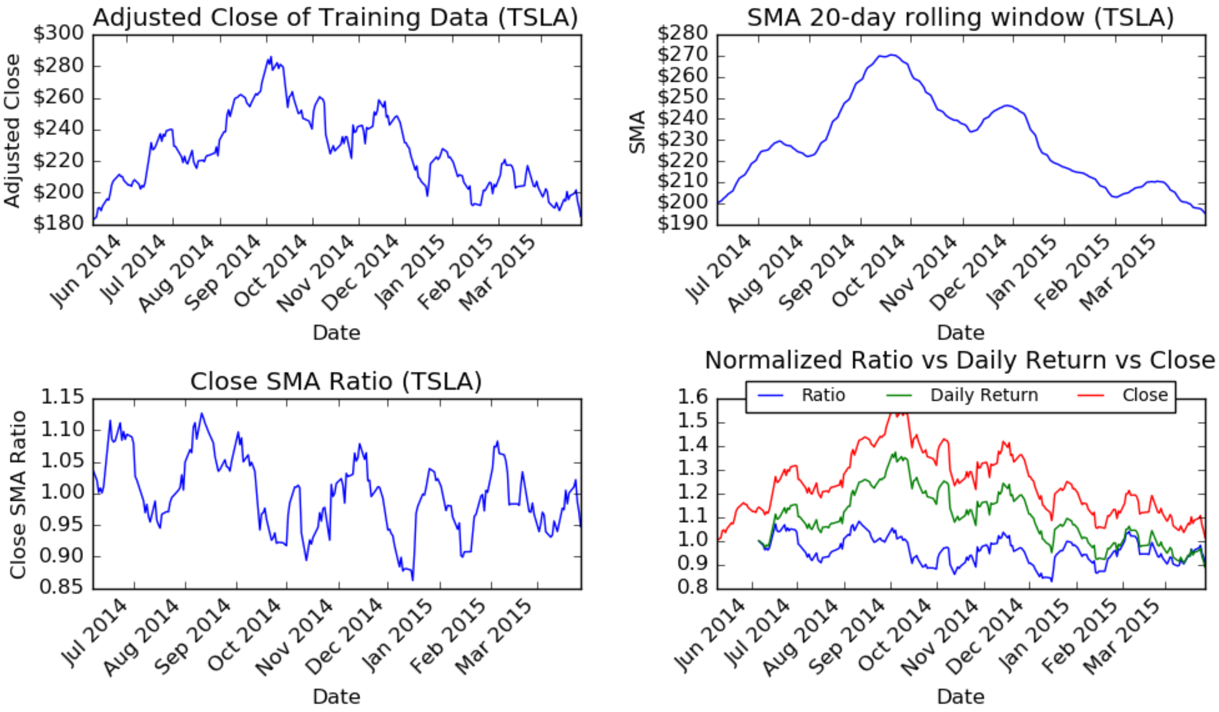


Fig 7. Various visualizations related to Close SMA Ratio.

The above plots underline the characteristics of Close SMA Ratio even further. Notice that most of the times the trend of normalized ratio is closer to daily returns when compared to close price that justifies the former's use.

2. Bollinger Bands (20)

Bollinger bands capture the volatility of the data by calculating two standard deviations above and below SMA. Similar to the above, the calculation of SMA used in Bollinger bands use 20-day rolling window.

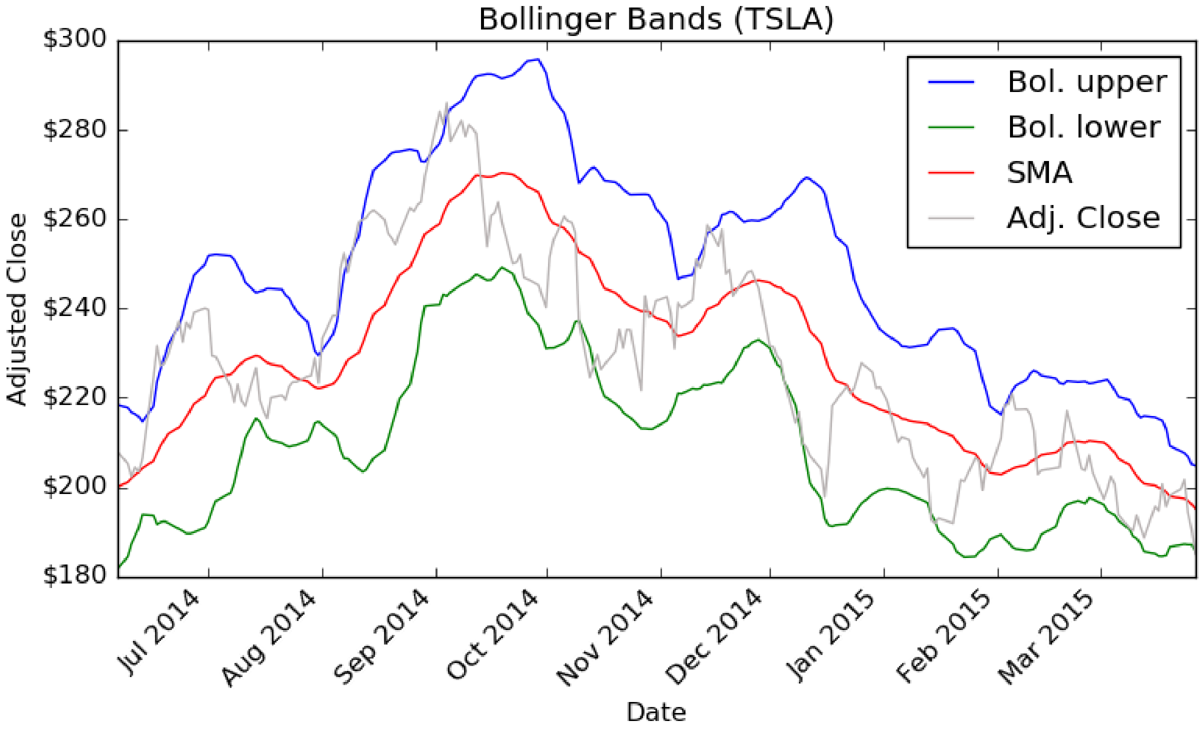


Fig 8. Bollinger bands calculated in this project.

The author included both upper and lower bands into the list of features used by the agent.

3. Daily Returns

Daily returns were calculated by measuring the differences between a day's adjusted close to its previous one. Refer to Figure 4 and 5 above for visualizations.

Implementation

The system uses Apache Spark hosted on a Databricks (http://databricks.com) community server to run its training and testing. It has the following modules:

1. Environment: The main entry point of the system. At the current version, it manages only a single trader, but later can be expanded to work with multiple traders, each monitors a different equity.
2. RLTrader: A trader object that learns using Q-learning algorithm. The object from this module stores the Q-table for a given equity.

3. Logger: An object that deals with logging debug messages and trial parameters. This object is attached to the Environment object
4. Simulator: This object simulates the training process of the algorithm. At the end of the simulation process, the trader should have a trained Q-table usable in testing.
5. There are also several helper and visualization-related functions to do various tasks in this project.

Output

This system outputs the following artifacts:

Q-table (CSV): Its content looks something like this:

```
,equity,state,hold,buy,sell
0,GOOG,90409090002,-5036.559082566398,-4778.897669791997,-3074.592960000009
1,GOOG,60806000002,-6438.282230513179,-6511.244911712901,-5208.637328133145
2,GOOG,10902070102,-3851.4499083486176,-4187.768959053048,-1507.8985908671252
3,GOOG,90208090502,-804.0998960793268,-782.2615811614439,1273.6294635847682
4,GOOG,90009090502,-947.3316645039924,-999.1226106822787,1340.4724628820165
```

Config file (JSON): This contains the configurations for Environment and RLTrader objects.

Trial log (Parquet): An Apache Spark Dataframe object that contains the report of all trials that was done during the training process. Each trial records the following items:

- trial: Trial number.
- training: True if trial was part of training.
- equity: Equity of this trial.
- start_time
- end_time
- times_profit: Number of times chosen actions were profitable i.e. reward > 0.
- times_draw: reward == 0
- times_loss: reward < 0
- total_rewards: Total rewards accumulated in this trial.
- total_portfolio: Total portfolio at the end of this trial.
- return_pct: Percent return compared to the beginning of this trial.
- parameters: JSON string containing learning rate, exploration rate, and discount factor.

Test log (Parquet): An Apache Spark Dataframe object that contains the report of all steps that was done during the testing process. Each step records the following items:

- time
- total_portfolio: Total portfolio at the end of this step.
- return_pct: Percent return compared to the beginning of this step.
- action: Action taken in this step. Could be "buy", "hold", or "sell".
- action_size: Number of shares affected by this action.
- close: Adjusted close price at the start

Step

During each step, the following processes occur:

1. Environment module creates an event from the data of the current step. Refer to the "Algorithms and Techniques" section above for what information a step entails.
2. It then creates a state from that event by extracting a selected set of features and discretize + combine them.
3. RLTrader module should then pick the optimal action based on a given state, otherwise randomly pick any action.
4. Right after moving to the next step, Environment module evaluates the action trader just picked in the above step. "Action Evaluation" section below explains this in more details.
5. Update the Q-table based on the evaluation result, and move to next step.

Action Evaluation

When evaluating an action, the definition of "profitable" is as follows for a long transaction ((for "short transactions," simply reverse the bolded words below):

- When a hold action was chosen, and the daily return for the next event was **positive**. Reward: $(r_{e+1} \times h_e) - c$ where r_{e+1} is the daily return of next event, h_e is the size of held stock in current event, and c is broker's commission (currently set to $0.0075 per trade with a minimum of $1.00).
- When a **buy** action was chosen, and the daily return for the next event was **positive**. Reward the agent with $r_{e+1} \times (h_{e+1} - h_e) - c$ in this case.
- When a **sell** action was chosen; with the reward formula of $(r_e \times h_e) - c$. The reasoning here is that the author did not want the trader to accumulate too much profit without **selling**, thus risking the investment.

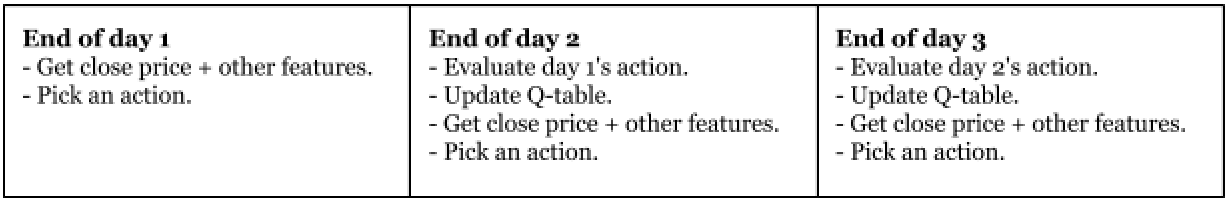
This project does not currently address the following complexities:

1. The possibility that the number of traded shares may not be exactly as the trader had requested.
2. How the order might have affected the price of that equity.

A slippage model needs to be implemented in the simulation to address the problems above. More on this under the "Improvement" section later.

Real-world Scenario

In practice, at the end of a trading day, closing trading data for that day will be gathered, and a new action will be chosen based on the agent's knowledge. On the next day, the agent should evaluate its previous day's action.



End of day 1	End of day 2	End of day 3
- Get close price + other features. - Pick an action.	- Evaluate day 1's action. - Update Q-table. - Get close price + other features. - Pick an action.	- Evaluate day 2's action. - Update Q-table. - Get close price + other features. - Pick an action.

Report Visualization

This project uses Python's matplotlib module for visualizing the report. This report shows the following information:

- Total returns of each trial. Useful to determine whether the the agent had properly learned.
- Number of profits, draws, and losses of each trial.
- Parameters in each trial (learning rate, exploration rate, discount factor).
- Summary statistics of both training and testing phases.
- Testing performance.
- Statistics of testing phase. It presents various useful metrics like Sharpe ratio, alpha, and beta.
- A scatterplot comparing the returns of the model and SPY.

Refinement

Before doing a full benchmark of the model, the author had tested some changes to the parameters as follows:

- Learning rate (α): Try out 0.1, 0.2, and 0.3.
- Epsilon decay function (ε): Try the following functions:

  $\epsilon = cos(0.05t)$, $\epsilon = a^t$, and $\varepsilon = \frac{1}{t^2}$
- Discount factor (γ): 0.05, 0.1, and 0.2.

For this initial benchmark, stock data of TSLA from date 2016-05-02 to 2016-09-14 were used with 20-day trailing window. The first 54 days (2016-05-31 to 2016-08-16) were used as training data and the last 19 days (2016-08-18 to 2016-09-14) as test data. This particular range was chosen somewhat arbitrarily, with the only requirement that the close prices of testing phase (Min: $194.47, Max: $225.00, Avg: $211.10) are between the close prices of training phase (Min: $193.15, Max: $241.80, Avg: $216.55).

The following tables report this initial benchmarking results in more details, with best values bolded (highest values for Sharpe and Alpha, and smallest absolute values for Beta). Note also that α (alpha) in these tables refers to learning rate instead of the metric Alpha.

Sharpe

α\ε	$\epsilon = cos(0.05t)$	$\epsilon = 0.05^t$	$\varepsilon = \frac{1}{t^2}$
0.1	-18.76/-19.95/-16.30	-19.96/-17.15/-20.86	-20.11/-23.68/-19.42
0.2	-19.36/-15.80/-18.44	-17.67/-21.57/-16.92	-19.64/-17.70/-20.32
0.3	-19.34/-18.49/-18.37	-17.46/-19.99/-19.92	-19.72/**-15.11**/-19.17

$\gamma = 0.05$ / $\gamma = 0.1$ / $\gamma = 0.2$

Alpha

α\ε	$\epsilon = cos(0.05t)$	$\epsilon = 0.05^t$	$\varepsilon = \frac{1}{t^2}$
0.1	-0.06/-0.03/-0.05	-0.04/-0.04/-0.03	-0.03/-0.03/-0.04
0.2	-0.07/**-0.01**/-0.05	-0.06/-0.04/-0.05	-0.03/**-0.01**/-0.03
0.3	**-0.01**/-0.04/-0.06	-0.05/-0.05/-0.04	**-0.01**/-0.03/-0.03

$\gamma = 0.05$ / $\gamma = 0.1$ / $\gamma = 0.2$

Beta

α\ε	$\epsilon = cos(0.05t)$	$\epsilon = 0.05^t$	$\varepsilon = \frac{1}{t^2}$
0.1	0.74/0.27/0.78	0.34/0.61/0.13	**-0.01**/-0.21/0.68
0.2	1.10/-0.14/0.47	1.09/0.45/0.39	0.20/-0.07/-0.08
0.3	-0.21/0.02/0.89	0.52/0.46/0.65	-0.24/0.11/-0.16

$\gamma = 0.05$ / $\gamma = 0.1$ / $\gamma = 0.2$

There are six best parameters candidates:

a. **S: -15.11** A: -0.03 B: 0.11 (α: 0.3, ε: 2, γ: 0.1)
b. S: -15.80 **A: -0.01** B: 0.14 (α: 0.2, ε: 1, γ: 0.1)
c. S: -19.34 **A: -0.01** B: -0.21 (α: 0.3, ε: 1, γ: 0.05)
d. S: -17.70 **A: -0.01** B: -0.07 (α: 0.2, ε: 3, γ: 0.1)
e. S: -19.72 **A: -0.01** B: -0.24 (α: 0.3, ε: 3, γ: 0.05)
f. S: -20.11 A: -0.03 **B: -0.01** (α: 0.1, ε: 3, γ: 0.05)

Recall that the "Metrics" section above mentioned that Alpha is more important than Sharpe ratio, and followed by Beta. Following that order of importance, the author chooses the setting of parameters of **candidate b** for the training phase of the final algorithm, that is:

Learning rate of 0.2, discount factor of 0.1, and Epsilon decay algorithm of $\epsilon = cos(0.05t)$.

The next section presents the results of the final algorithm with the above optimal settings.

Results

Model Evaluation and Validation

As explained in the "Problem Statement" section above, in **scenario 1** the trader randomly picks an action, in **scenario 2** the trader stops learning in testing phase, and in **scenario 3** the trader keeps learning with smaller learning rate and discount factor in testing phase.

Appendix I shows the reports of various scenarios. In figure 9, notice that Total Returns and numbers of profits, draws, and loss do not show any improvement over more trials. Contrast this with the improving trends illustrated in figure 10 and 11.

As it turned out, at least for the given dataset and dates, the third scenario ended up having a higher Sharpe ratio, higher Alpha, and closer to zero Beta. Although this scenario had negative total returns, all these metrics suggested that it was arguably a better setting.

Two more scenarios were created to investigate this further:

- **Scenario 4**: 65 training days, the trader **does not learn** in its testing phase.
- **Scenario 5**: 65 training days, the trader **learns** in its testing phase.

Scenario	Train days	Test days	Sharpe	Alpha	Beta	Returns %
1	183	19	-11.55	-0.02	-1.12	-3.85%
2	183	19	-6.71	-0.01	-0.79	-2.57%
3	183	19	-17.29	-0.03	-0.73	-6.20%
4	65	19	19.90	0.03	-0.03	6.18%
5	65	19	-23.89	-0.03	-0.71	-5.20%

Justification

We can observe the following trends from the table above:

- The performance of Q-learning powered trader (scenario 2 - 4) does not seem to improve compared to the random walk agent. One could argue, however, that Q-learning reduces the volatility of the portfolio, as seen from their Beta values that consistently smaller than the random agent.

- The effect of reducing the training days is still unknown, as the performance seemed to increase in scenario 2 and 4, but it did decrease in scenario 3 and 5.
- From these cases, stopping the agent from learning yielded better results (i.e. higher Alpha and Sharpe ratio). This trend occurred in both shorter and longer training phases.

This final solution is however not conclusive just yet. The author had only tested the trained model with 19 days of testing data, and it was very possible that these results were simply coming from a random selection. We also have yet to see whether this system would work over a longer trading period.

Conclusion

Reflection

To summarize, the experiments conducted had found out that:

- The model where the agent learns had performed better than a random walk.
- During testing phase or in a real-world situation, it is better to have the agent to keep incorporating learning algorithm instead of only using what it has previously learned.

The most interesting aspect of this project for the author personally was how with a relatively simple algorithm the agent was able to choose optimal actions.

Improvement

Before we can deploy this system in a real-world market, there are still numerous experiments and adjustments to the algorithm that are still required. This section explains an overview to some of them.

Longer Experiment Period

To see whether the system would work for a longer trading period, some adjustments to the Simulation module are still needed.

The current simulation simulates only a portion of the entire trading period and uses the learned trader to predict a much smaller portion in the future. A better simulation would be to extend this into a longer period, possibly by shifting forward the training data.

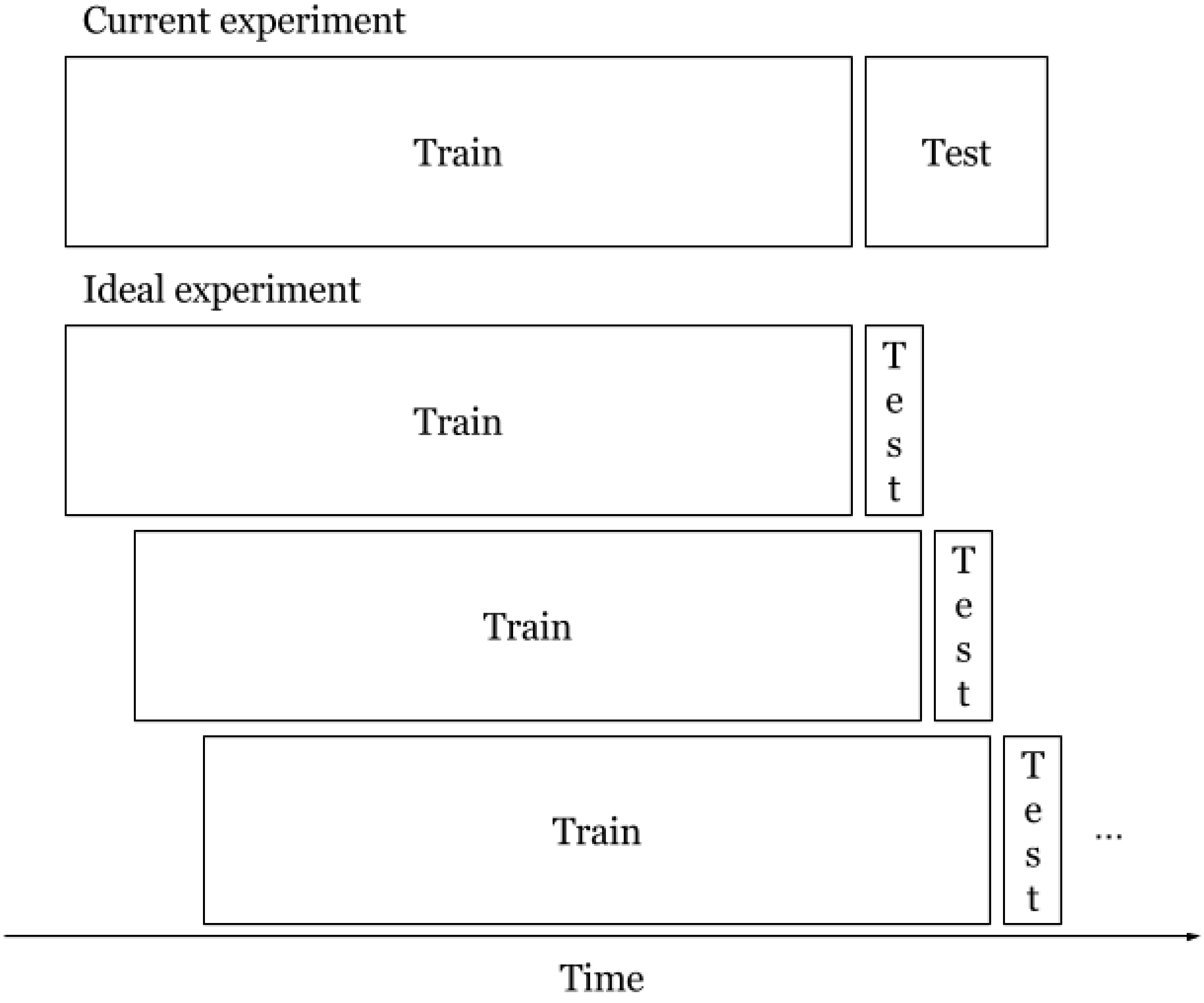


Fig 14. Visualizing current experiment vs. a longer ideal experiment.

Multiple Traders with Capital Asset Pricing Model

The current model is terribly risky as the trader puts in its entire capital on each trade. We can calculate CAPM when trading to decide how much percentage of capital should the trader use on each equity. This system will require the system to have multiple agents, each monitoring a different equity simultaneously. The asset allocation from a CAPM calculation ensures the system divides the portfolio optimally across multiple industries, which lowers its covariance i.e. when an industry is bearish another bullish industry can offset its losses.

Parallel Computing

One possible way to work with multiple traders is by using Apache Spark's parallelization. The author is not entirely sure yet how to implement this at the time of writing. The challenge is that Q-table needs to be updated iteratively as the system also needs to examine past Q-values. This iterative update means that parallelization at the level of Q-table may not be possible.

Parallelization can probably be done in the traders' level e.g. by dividing the data by equities, let each trader handle a single equity, then recombine the predictions into a CAPM calculation. Figure 14 shows this in more detail.

It is also possible for the system to train in parallel, although it is quite challenging to do. Another thing we could do instead is to perform multiple training phases in parallel, then combine the resulting Q-table.

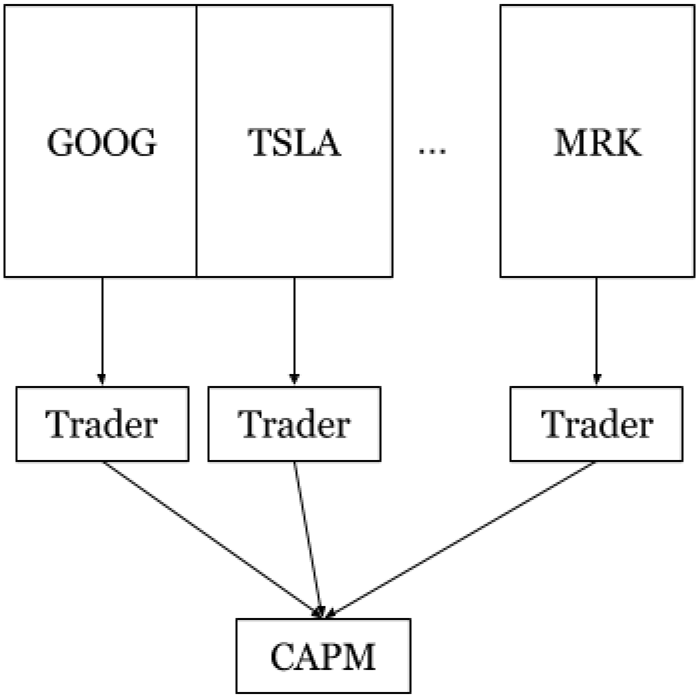


Fig 15. Separate processes for Multiple traders followed by a CAPM calculation.

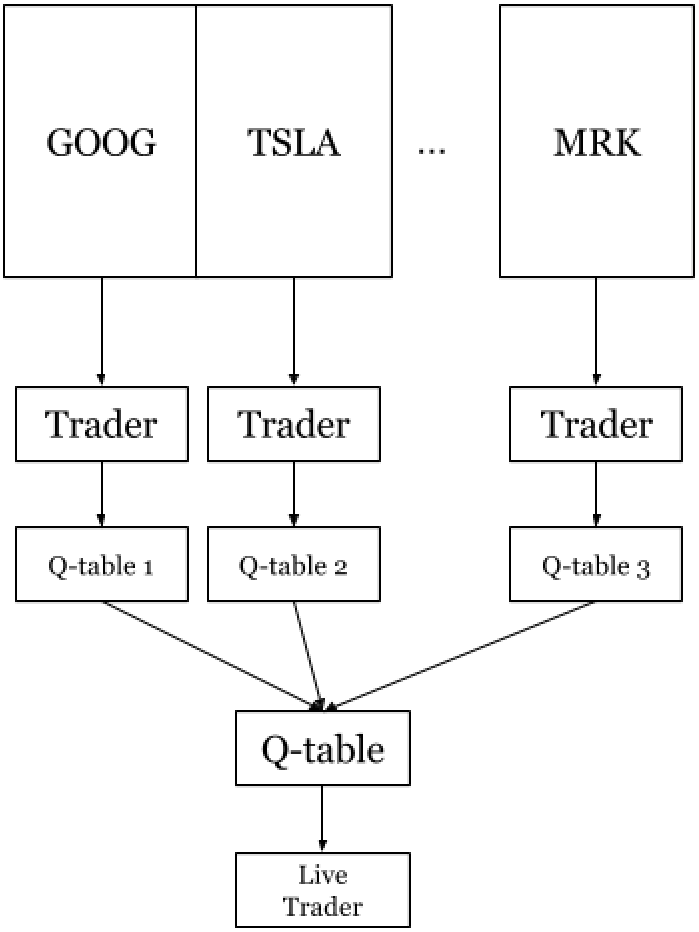


Fig 16. Multiple traders create several Q-tables which can later be combined and used by the Live Trader (the one operating in Quantopian server).

Find The Optimal Rolling Window

"Data Preprocessing" section above mentioned that methods such as Akaike Information Criterion (AIC) or Bayesian Information Criterion (BIC) could be implemented to find the optimal number of the rolling window.

Both AIC and BIC are metrics we can use to measure the quality of a model relative to other models.

A simple moving average (SMA) model is represented as follows:

$$y_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \ldots + \theta_p \varepsilon_{t-p}$$

Where μ is the mean of the series, $\theta_1, \ldots, \theta_p$ are the parameters of the model, and $\varepsilon_1, \ldots, \varepsilon_p$ are white noise. For our purpose, we will consider the mean to be 0.0 (i.e. normalized):

$$y_t = \varepsilon_t + \theta_1 \varepsilon_{t-1} + \ldots + \theta_p \varepsilon_{t-p}$$

The best value of p can then be found by comparing AIC or BIC scores for several SMA models. What makes this possible is because both AIC and BIC use the number of estimated parameters in their scores; for example, the representation of AIC is as follows:

$$AIC = 2k - 2ln(\hat{L})$$

where

- $\hat{L} =$ the maximized value of the likelihood function of the model M, i.e. $\hat{L} = p(x|\hat{\theta}, M)$, where $\hat{\theta}$ are the parameter values that maximize the likelihood function;
- $x =$ the observed data;
- $k =$ the number of free parameters to be estimated (including intercept for a linear model). In the SMA model above, this should be $p + 1$.

Implement a Slippage Model

The fact that the algorithm does not currently take slippage into consideration could potentially cause a problem. That is, the price the algorithm decided to trade at is not necessarily the exact execution price, nor is the number of shares traded. In fact, the reward and total portfolio calculation both use closing price, which is not correct since the trade would only happen the next day.

A more realistic model can be made by implementing a slippage model into the algorithm.

From Wikipedia:

> To properly understand slippage, let's use the following example: Say, you (as a trader) wanted to purchase 20,000 shares of SPY right now. The problem here is that the current ASK price of $151.08 only contains 3900 shares being offered for sale, but you want to purchase 20,000 shares. If you need to purchase those shares now, then you must use a market order and you will incur slippage by doing so. Using a market order to purchase your 20,000 shares would yield the following executions (assuming no hidden orders in the market depth):
>
> Buy 2800 @ $151.08
> Buy 1100 @ $151.08
> Buy 3800 @ $151.09
> Buy 900 @ $151.10
> Buy 3700 @ $151.11
> Buy 1200 @ $151.12

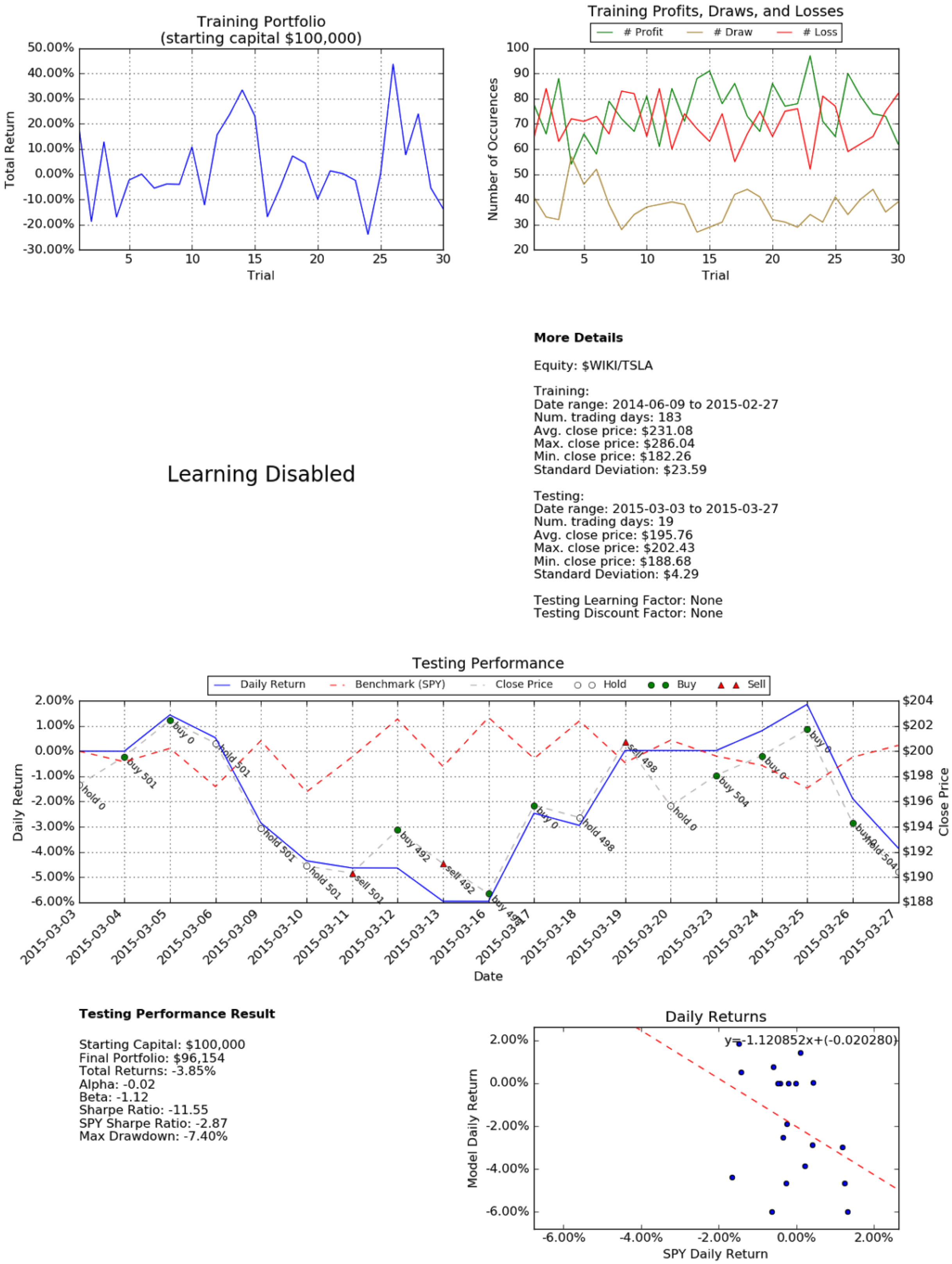
Buy 3700 @ $151.13
Buy 200 @ $151.14
Buy 1000 @ $151.15
Buy 400 @ $151.18
Buy 100 @ $151.22
Buy 600 @ $151.24
Buy 500 @ $151.25 (only 500 shares out of the 2000 being offered at this print point are executed, because this will represent our entire 20,000 share order).

News-aware Trader

One could argue that the current feature selection does not capture all the information needed to beat the market consistently, and that would be correct. Another interesting area of research is to incorporate deep learning methods such as LSTM to allow the reader to "extract" market knowledge from news sites (also Mr. Trump's tweets, perhaps?) and include them in the learning process.

Appendix I. Reports of all 5 scenarios

Next 5 pages show the reports of all 5 scenarios.



More Details

Equity: $WIKI/TSLA

Training:
Date range: 2014-06-09 to 2015-02-27
Num. trading days: 183
Avg. close price: $231.08
Max. close price: $286.04
Min. close price: $182.26
Standard Deviation: $23.59

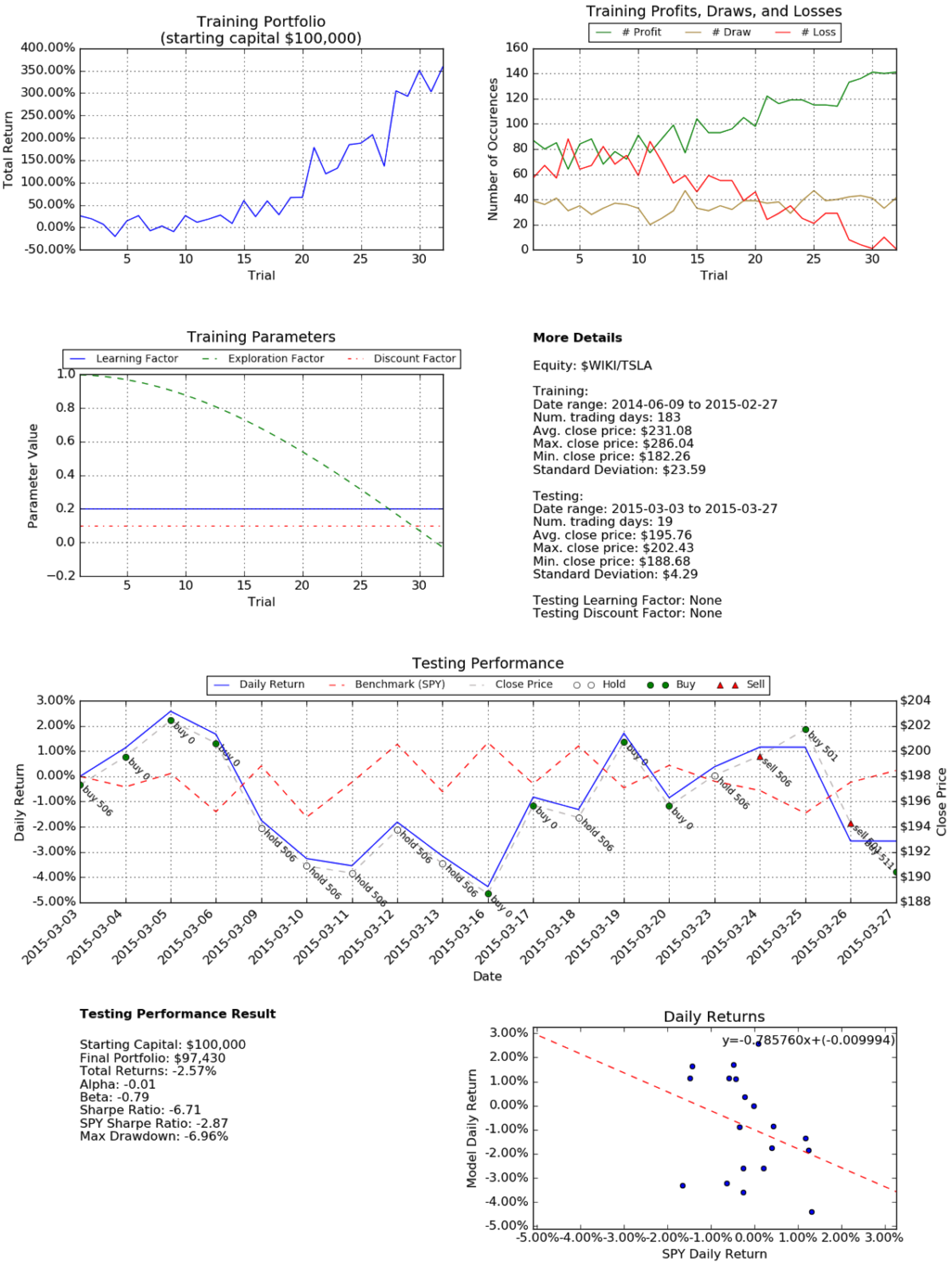
Testing:
Date range: 2015-03-03 to 2015-03-27
Num. trading days: 19
Avg. close price: $195.76
Max. close price: $202.43
Min. close price: $188.68
Standard Deviation: $4.29

Testing Learning Factor: None
Testing Discount Factor: None

Testing Performance Result

Starting Capital: $100,000
Final Portfolio: $96,154
Total Returns: -3.85%
Alpha: -0.02
Beta: -1.12
Sharpe Ratio: -11.55
SPY Sharpe Ratio: -2.87
Max Drawdown: -7.40%

Fig 9. Result of scenario 1.



More Details

Equity: $WIKI/TSLA

Training:
Date range: 2014-06-09 to 2015-02-27
Num. trading days: 183
Avg. close price: $231.08
Max. close price: $286.04
Min. close price: $182.26
Standard Deviation: $23.59

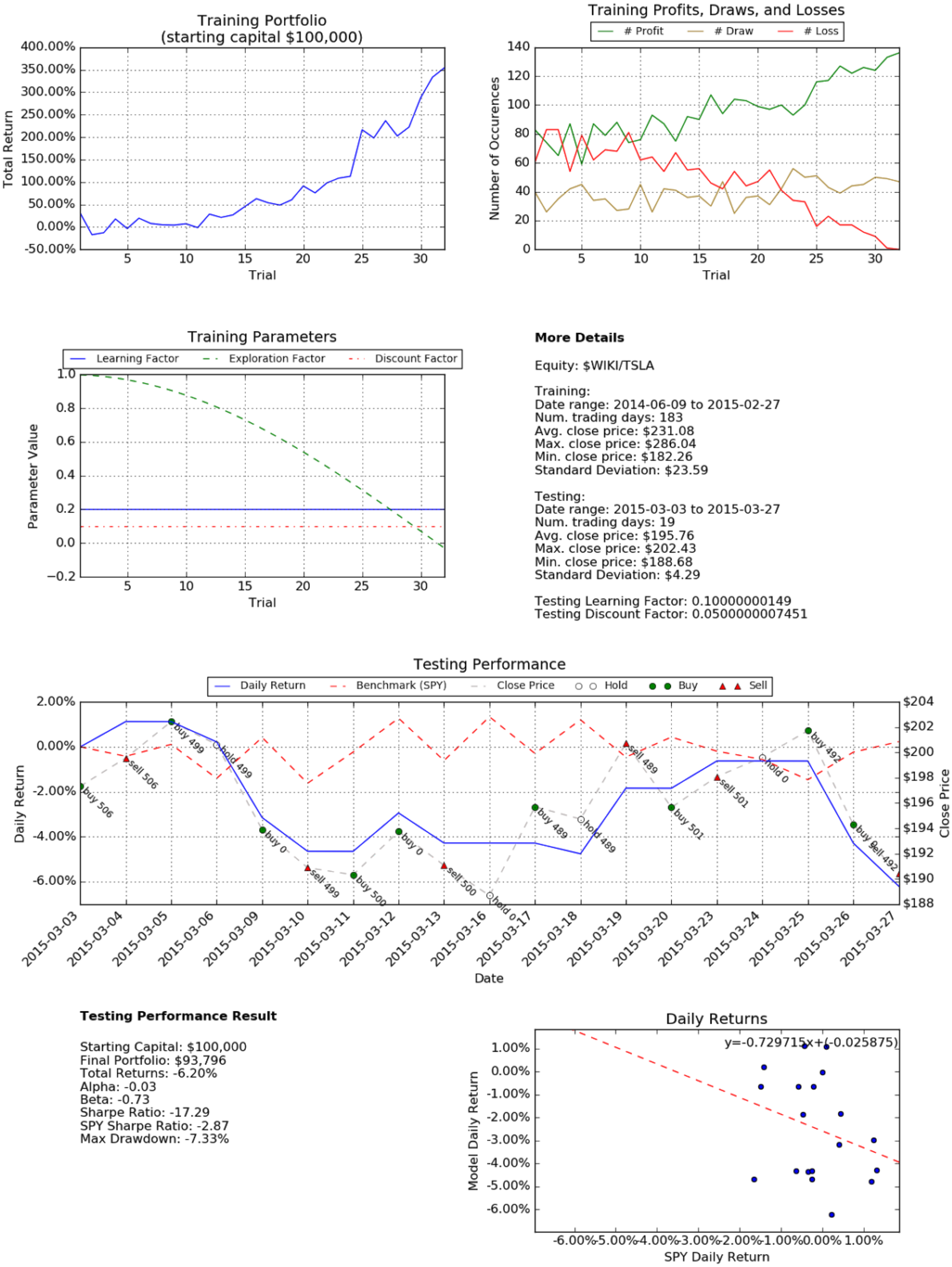
Testing:
Date range: 2015-03-03 to 2015-03-27
Num. trading days: 19
Avg. close price: $195.76
Max. close price: $202.43
Min. close price: $188.68
Standard Deviation: $4.29

Testing Learning Factor: None
Testing Discount Factor: None

Testing Performance Result

Starting Capital: $100,000
Final Portfolio: $97,430
Total Returns: -2.57%
Alpha: -0.01
Beta: -0.79
Sharpe Ratio: -6.71
SPY Sharpe Ratio: -2.87
Max Drawdown: -6.96%

Fig 10. Result of scenario 2.



More Details

Equity: $WIKI/TSLA

Training:
Date range: 2014-06-09 to 2015-02-27
Num. trading days: 183
Avg. close price: $231.08
Max. close price: $286.04
Min. close price: $182.26
Standard Deviation: $23.59

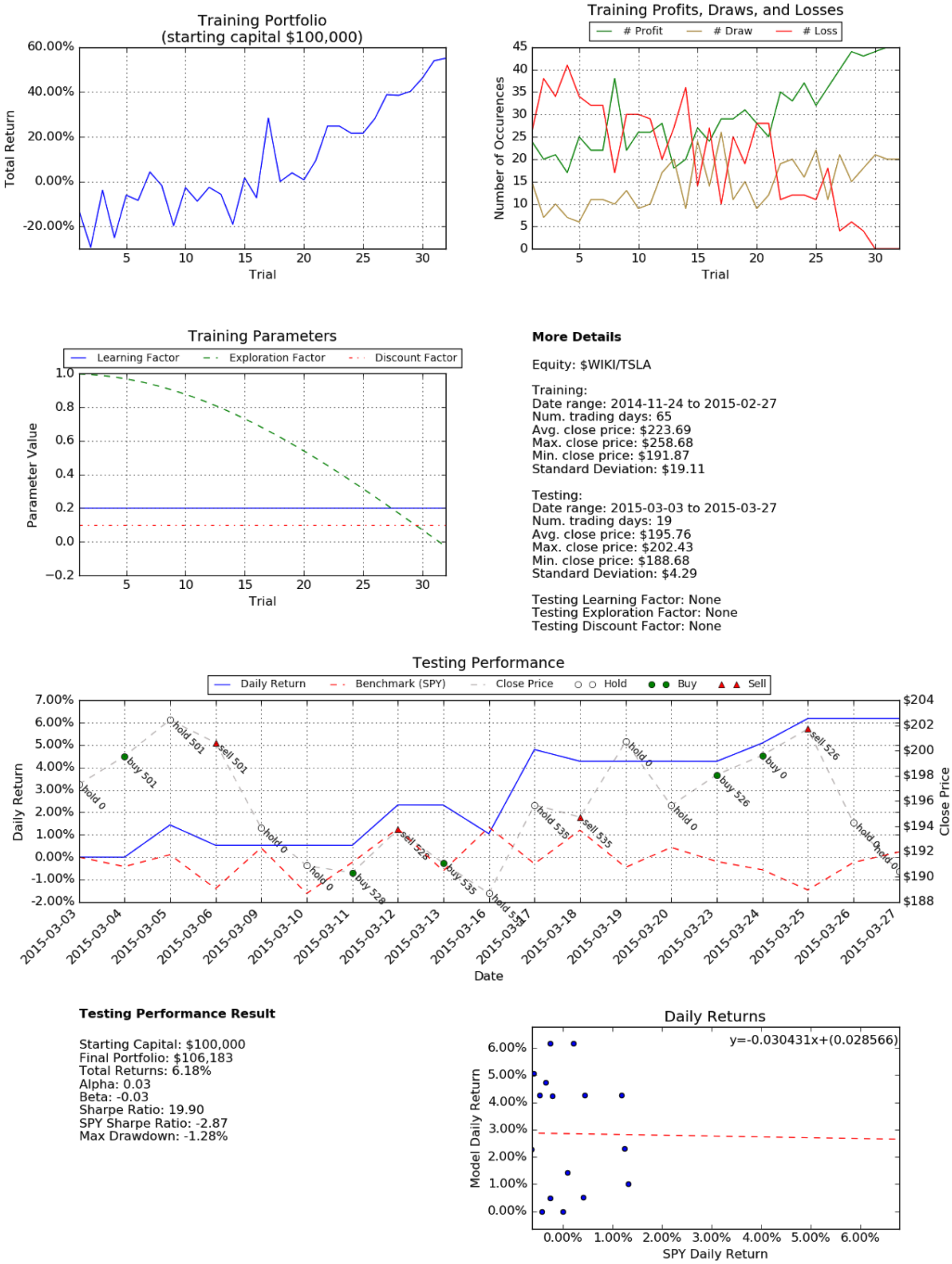
Testing:
Date range: 2015-03-03 to 2015-03-27
Num. trading days: 19
Avg. close price: $195.76
Max. close price: $202.43
Min. close price: $188.68
Standard Deviation: $4.29

Testing Learning Factor: 0.10000000149
Testing Discount Factor: 0.0500000007451

Testing Performance Result

Starting Capital: $100,000
Final Portfolio: $93,796
Total Returns: -6.20%
Alpha: -0.03
Beta: -0.73
Sharpe Ratio: -17.29
SPY Sharpe Ratio: -2.87
Max Drawdown: -7.33%

Fig 11. Result of scenario 3.



More Details

Equity: $WIKI/TSLA

Training:
Date range: 2014-11-24 to 2015-02-27
Num. trading days: 65
Avg. close price: $223.69
Max. close price: $258.68
Min. close price: $191.87
Standard Deviation: $19.11

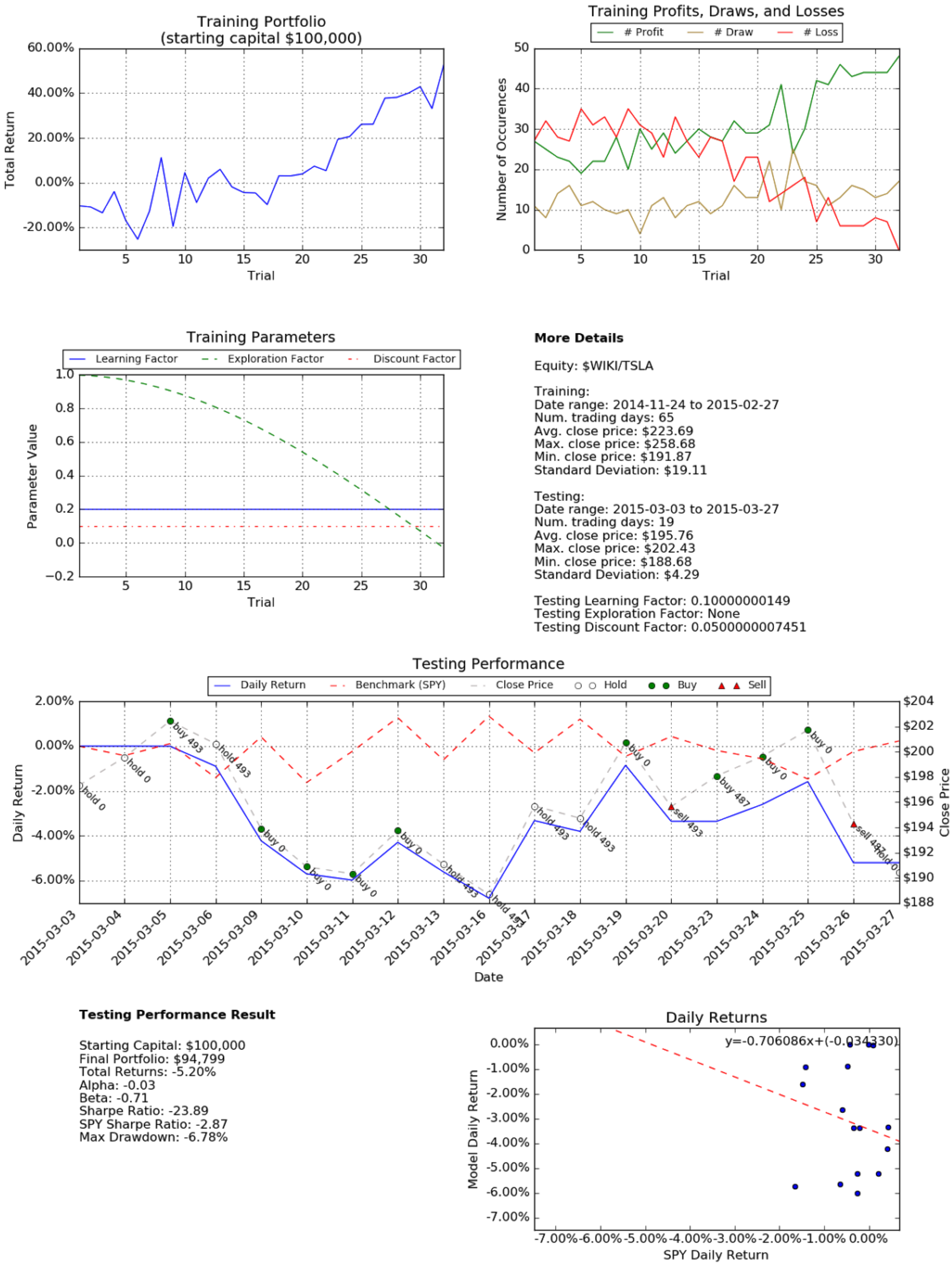
Testing:
Date range: 2015-03-03 to 2015-03-27
Num. trading days: 19
Avg. close price: $195.76
Max. close price: $202.43
Min. close price: $188.68
Standard Deviation: $4.29

Testing Learning Factor: None
Testing Exploration Factor: None
Testing Discount Factor: None

Testing Performance Result

Starting Capital: $100,000
Final Portfolio: $106,183
Total Returns: 6.18%
Alpha: 0.03
Beta: -0.03
Sharpe Ratio: 19.90
SPY Sharpe Ratio: -2.87
Max Drawdown: -1.28%

Fig 12. Result of scenario 4.



More Details

Equity: $WIKI/TSLA

Training:
Date range: 2014-11-24 to 2015-02-27
Num. trading days: 65
Avg. close price: $223.69
Max. close price: $258.68
Min. close price: $191.87
Standard Deviation: $19.11

Testing:
Date range: 2015-03-03 to 2015-03-27
Num. trading days: 19
Avg. close price: $195.76
Max. close price: $202.43
Min. close price: $188.68
Standard Deviation: $4.29

Testing Learning Factor: 0.10000000149
Testing Exploration Factor: None
Testing Discount Factor: 0.0500000007451

Testing Performance Result

Starting Capital: $100,000
Final Portfolio: $94,799
Total Returns: -5.20%
Alpha: -0.03
Beta: -0.71
Sharpe Ratio: -23.89
SPY Sharpe Ratio: -2.87
Max Drawdown: -6.78%

Fig 13. Result of scenario 5.

References

https://en.wikipedia.org/wiki/Moving_average
http://www.investopedia.com/ask/answers/102714/whats-difference-between-alpha-and-beta.asp
https://en.wikipedia.org/wiki/Akaike_information_criterion
Various lectures in Udacity from *Machine Learning for Trading* and *Reinforcement Learning* courses.
https://en.wikipedia.org/wiki/Slippage_(finance)