Long Nguyen
Udacity
Machine Learning Engineer
Capstone project
June 11, 2016

Machine learning for Investment

Definition

Project overview

This project is done as fulfillment for capstone project of Machine Learning Engineer Nano degree at udacity.com. The chosen subject is the application of machine learning techniques in the investment and domain trading. The machine learning techniques include supervised learning, unsupervised learning, and reinforcement learning.
The dataset for this project are the stock price of large public trading corporations downloaded from Yahoo finance historical data.

The data can be divided into the following categories:

- Companies that have shown huge growth over long period of time, for examples Google, Apple, Intel, Microsoft IBM.
- Tech "stars" like Facebook and Tesla.
- Large (industrial/ healthcare) technology companies that had sluggish grow in the last couple of years, for examples Philips, GE, Siemens.
- Companies in decline, for examples Yahoo and HP.
- Non-tech companies, for examples Exxon, Shell, Bank of America, McDonald, WalMart.

Each company historical stock data is downloaded as a separate .csv file. Each row in the data indicate the price of one trading day: Open (Price), High (Price), Close (Price), (Total Traded) Volume, and Adjusted Close price (adjusted for stock split).

The implementation makes use of Python and standard Python machine learning packages: numpy, pandas, sckitlearn and matplotlib.
All collected data are stored in Data sub folder relative to the python code file. File name is the company name.

Problem Statement

This project intends to look at these dataset to draw learning and knowledge from 2 different angles:

1. Can we use Open Price, High Price, Close Price, Volume, and maybe some derived data (called features from now on) to predict Adjusted Close (call target from now on)?
2. Can we train a "Trading agent" to learn from the historical data to it can make independent trading decisions to maximize profit overtime?

(1) is a classic regression supervised learning problem: from a number of feature columns we need to train our learning model to fit the data and predict the feature (Adj Close). Different techniques have been taught and applied during the course of Udacity Machine Learning Nanodegree program: Gradient Boosting, Decision Tree regression, Support Vector Machine (SVM) regression… Gradient Boosting is technique that is widely used among academics and practitioner because of its effectiveness and high prediction performance. In this project Gradient Boosting is used and optimized with GridSearch. A simple implementation of Ada boost and decision tree regressor are also tried out for comparison.

The expectation is our models shall fair well since the input data and output data are highly correlated and deducible: e.g. from Close to Adj Close is a question of adjustment for stock split. Also when we can provide a "Rolling average of Adj close from previous days", the expectation is the Adj close of the new day shall bounce closer to the mean.

(2) The question mentioned above is an interesting academics one but provides little practical values and usage: it is of course interesting to see how well the supervised learner/regressor predicts the target data from feature data, but this provides little values for trading.

It would be very interesting to see if we can feed a reinforcement learning agent the historical data, and allow him to learn and make independent trading decisions (sell, buy, hold). For our "smart" trading agent, a Q-learning algorithm is implemented.

As a side not, it is really tough to beat this "native" human trader if he choose the right stocks and put all his money on it 30 years ago (say bought Apple or Microsoft 30 years ago, or just buy a "star" stock). Chance for the Trader Agent is more in a shorter period or for stocks that have more moderate growth, decline, or volatility.

Most of the analysis and discussions in this report shall focus more on (2). (1) shall only be discussed briefly to satisfy academics curiosity.

Metrics

For (1) Mean Squared Error (MSE) between predicted Adj Close and actual Adj Close to judge the model performance. MSE is a robust and widely use performance metrics for regression model and is also suitable for stock price prediction.

For (2), to judge how effective the agent perform, we can compare the performance to "market gain" (say, if an "inattentive" human trader just buy a stock with all the money she/he have, and leave it there for years, can the stock agent with complex decision making process beat him?

The "market gain" can be calculated as the ratio of "rolling average" of Adj close at the end and begin of a time interval.

The "trader investor again" can be calculated as the ratio of total assets owned (stocks + available money) and asset at the beginning (capital given to do investment).

Analysis

Data Exploration

As mentioned earlier, data is downloaded from Yahoo finance historical data. The data format and semantics are standard and widely used among academics and practitioners.

There are a couple of remarks for these data:

1. Data is stored in reversed chronological order (latest data first). To visualize and analyze the data overtime, it is best to reverse the data.
2. Not all value for Date are present. This is not a big issue since the order is correct and date is not used for prediction.
3. As a side note for analyzing stock data: there is a dot com burst after 2000 and the great recession around 2008-2010.
4. Apart from date, data is in numerical values and no transformation is needed. For certain regression analysis, it would make sense to to scaling on Volume data (since this is in orders of magnitude larger than price). However, since the focus of this assignment is more on Q-learning, this is not done. Furthermore, when rolling average is used, Volume data has much less predictive value.
5. Adj Close has correction for stock split.

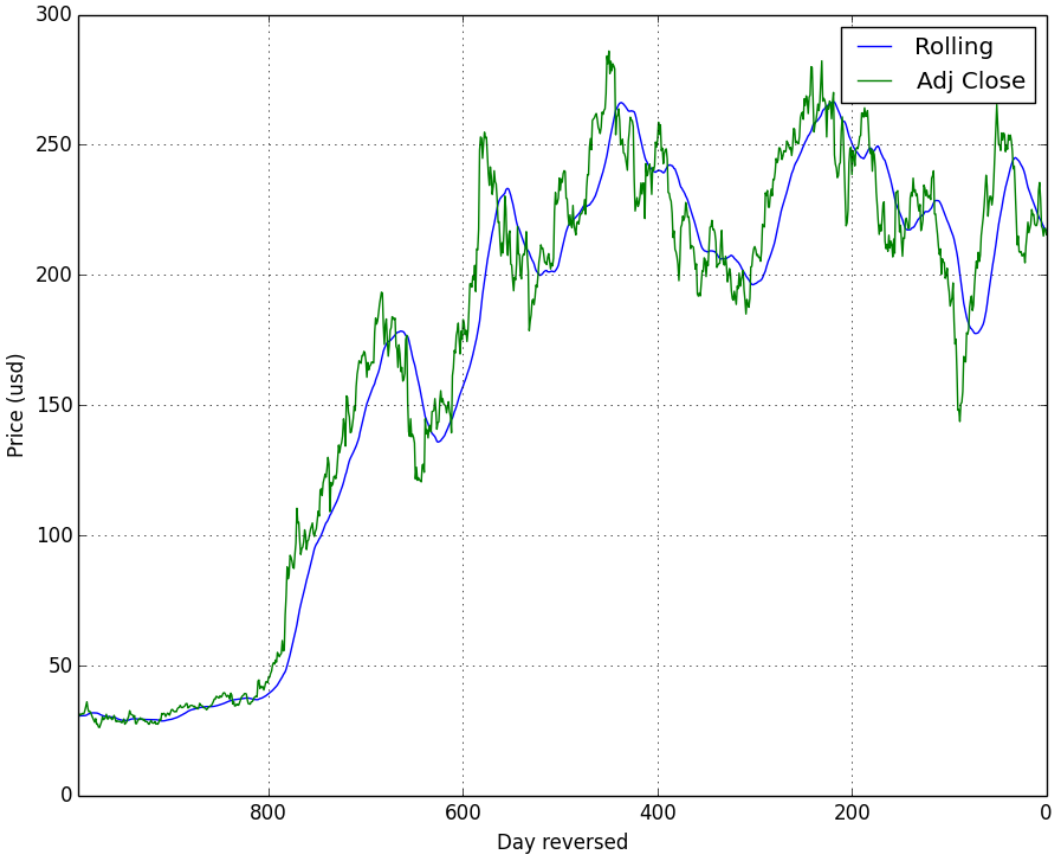
Standard statistical value can be calculated by using DataFrame.describe(). However, for this context, there will not be much value since we analyze a large number of stocks. Also for this assignment the most important data is price over time.

As shall be discussed later in more details, the provided data is “enhanced” with the following features:

1. Rolling mean of Adjusted close (of previous days). This is used both for Regression and Q-learning. Rolling mean shall smooth the idea and eliminate “outliers” (turbulence in prices)
2. Bollingen bands (lower and upper standard deviation of rolling mean). This is used for Q-learning. The rationale shall be explained later.

Exploratory visualization

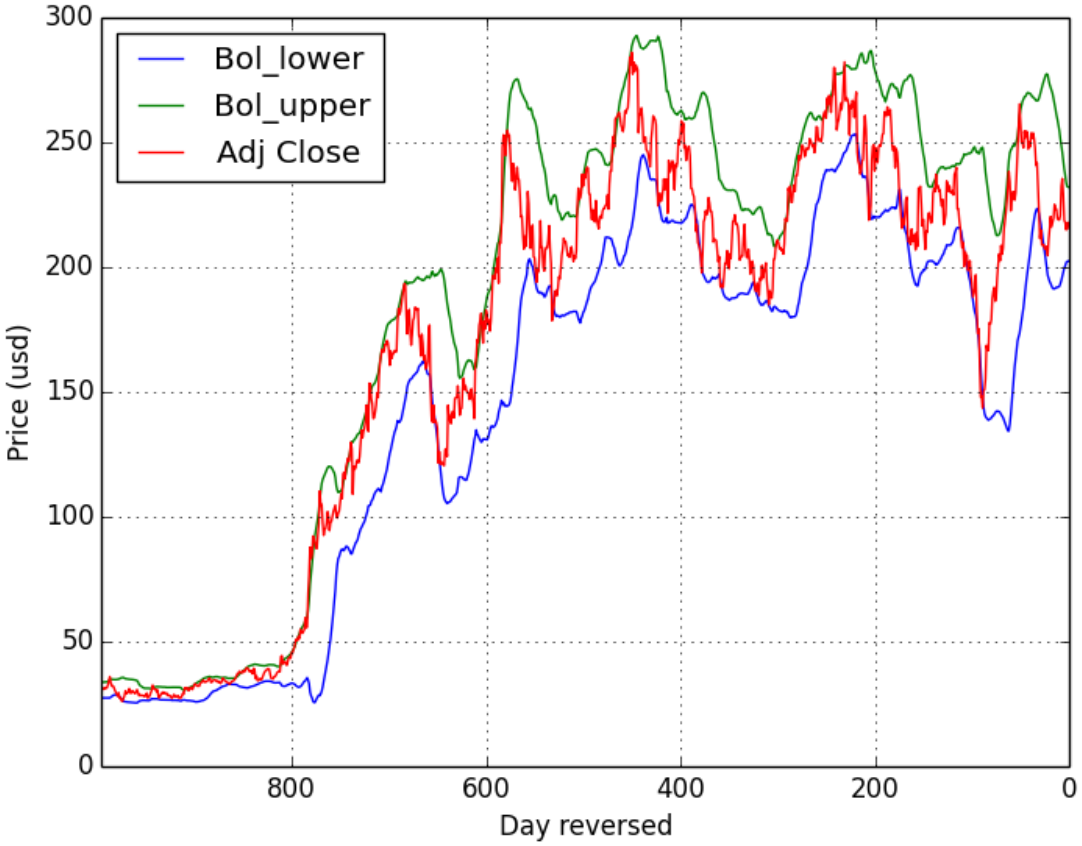
The figure below shows the Adj Close and Rolling mean of Adj Close (time window = 30 days) for Tesla. It can be observed that the rolling mean smooth out the Adj Close value (less edges). Another important fact to observe is the Rolling mean is “slower” in reflecting trend, meaning decrease/increase is stock shall be observed after the time window in the rolling mean.



The figure below shows the Adj Close versus Bollinger bands (lower and upper). Bollinger bands are calculated as 2 standard deviations from the rolling mean.

Observing the relation between Adj Close and lower and upper Bollinger band represent an excellent decision point for selling and buying stock:

1. When the Adjusted close price fall below/around the lower band, it is generally very cheap, and the price tends to increase after that (fall back into mean).
2. On the other hand, when the Adjusted close price fall above/ around the upper band, it is generally on the expensive side, and thus the price tends to decrease after that. (fall back after mean).



Algorithms and Techniques

As mentioned earlier, to solve the first question (predicting Adj Close from the feature column), Gradient Boosting regression is employed. Gradient Boosting is one of the highest performing supervised learning algorithms. The standard steps for the supervised machine learning using regression are:

1. Data preparation (including preprocessing, cleaning, enhancement if necessary).
2. Split data into train set (typically 60-70%) and test set to prevent model over fitting.
3. Define performance metrics. As discussed, Mean Squared Error (MSE) shall be used.
4. Create regressor (Gradient Boosting Regressor) (with standard / preferred parameters)
5. Fit regressor with train data set. Measure performance.
6. Use regressor to predict test data set, or other test set (e.g. newer stock data of the same symbol).
7. Use GridSearch to do a brute force through different parameter ranges to find parameters which give the best possible performance using metrics defined in (3).
8. Measure performance of the learner and optimize if necessary.

For our Trader agent using Q-learning, the following steps are required:

1. Define State, Action, and Reward (to be discussed in details in the implementation section).
2. Initialize Q-table with appropriate values (depending on domain and experience)
3. Iterate (until converge or until time/data is up)
    1. Find the best action to take from current state.
    2. Take the action, update state.
    3. Update Q table.
4. Measure and validate performance of the Q-learner.

Benchmark

As discussed earlier:

1. For regressor: the performance benchmark is obtained by comparing the predicted Adj Close and the real Adj Close of the test data set.
2. For Q-learner trading agent: the performance benchmark is the total current assets (capital + value of stocks) versus the capital given at the beginning.

Methodology

Data Preprocessing

The data set chosen in this project has a limited number of features (6) and limited in size (data collected in days instead of minutes or seconds), so no feature transformation (like Principle Component Analysis is used). There are no evident correlation among the feature columns.

The data is also properly cleaned up and all numeric so no data transformation is needed (date is not all filled in but not used).

However, we would like to enhance our data set with the following features:

1. Rolling mean of Adjusted Close: Rolling mean is widely seen by many financial expert as a great way to predict stock data (except for very short term investment). This is because everything tends to fall back into the mean (like gravitation). Rolling mean is used for both Gradient Boosting regressor and Q-learner. For the regressor it can be felt like this is cheating since we are using the mean of the target column to predict the target. But looking further it is not because:
    1. We calculate the mean by averaging “other” target, not the current target we want to predict (by shifting the window).
    2. This is a valid in practice because the rolling average of adjusted close of previous days are always available when trying to predict stock price of the “new day”.
2. Bollinger band: as discussed earlier, Bollinger bands are a great way to predict when the stocks are “relatively” cheap or expensive. This is used for the Q-learning trader.

Implementation

Implementation follows the steps discussed in “Algorithms and techniques” section. The implementation should be self-explanatory. Comments have been added when more explanation is required.

A couple of pointers:

1. trader_agent.py defines the class TraderAgent, which contains the implementation of the Q-learning trading agent.
2. Q_trader.py: does data reading, pre-process data, create and invoke TraderAgent.
3. Trader_regressor.py implement data preparation, pre-processing, fitting and validation of GradientBoostingRegressor. Most of GradientBoostingRegressor visualization is taken from (which some addition and adaptation, including GridSearch).

Use Q_trader.py or Trader_regressor.py independently as an entry point. These 2 are independent of each other.

These are the considerations for Q-learning implementation (some inputs and advices from Machine Learning for Trading course @Udacity):

1. Action is either “Buy”, “Sell”, or “Hold”. This is self-explanatory. The amount of stocks to buy or sell is not a part of the action (otherwise the Q-table shall be huge and will never converge). The amount of stocks to buy and sell are the “tuning parameters”. As an improvement this should be dependent on the short or long term trading or the volatility of the market.
2. Reward is the daily return after taking the action. Daily return should be calculated using Open and Close price (because it is specific for that day), not the Adjusted close.
3. State consists of the following information.
    1. Ratio of Adjusted Close over rolling mean. This evidently gives a hint on whether the stock shall likely to increase or decrease, which is really important for our “state”.
    2. Ratio of Adjusted Close over lower and upper Bollinger band: as explained earlier, this also give a hint on whether the stock price shall increase and decrease. Therefore it should be a part of the state information.

In the course “Machine Learning for trading”, it is recommended to use the return entry as part of the state. This has been tried out and decided not to be used because: (1) “sunk cost” is not really important in predicting the stock price. (2) this value is difficult to normalized or quantized. For example if we observe Microsoft/Apple price in the period of 30-40 years, the range can be between 0 and 700. When we observe a less performing stock, a typical range would only be 0 to 2 (very optimistic). This information should instead be used to tune or optimize other Q-learning parameters, for example to balance between exploration (random try out, e.g. if the return is already very good, we can be a bit more “risky”) versus exploitation (use the best action in the Q table).

All these information are “quantized” in a resolution range (it is currently selected to be in the range of 0..5. Given the 3 pieces of states and 3 actions, the size of Q-table is: 6 * 6 * 6 * 3 = 648.
This is a reasonable size, given the fact we typically have > 10 years of data to learn (~2700 entries). Furthermore, we can learn the same piece of data over and over again to optimize the Q table further.

4. Correction for splitsing: to calculate the current asset correct, the number of holding stocks should be used. This should be corrected with stock split information.
5. default Q-learning value: It is chosen to have (state, action = ‘buy’) with higher default Q-value than other actions since there is money to spend.

Refinement

For GradientBoostingRegressor, a GridSearch is performed to find the optimized parameter. Since this is not the focus of this project and GridSearch is a relatively simple exercise, this is not discussed further. The found optimized parameters are used in the final version. GridSearch does find parameters which give better performance than default parameters, but the difference is relatively small because the standard version already have a very good performance.

For the Q-learning trader implementation, the parameters refinement process is extensive:

1. As discussed above, default Q-learning value needs to be chosen properly, depending on money available and investment profile.
2. The amount of stocks to buy/sell needs to be tuned to get better performance.

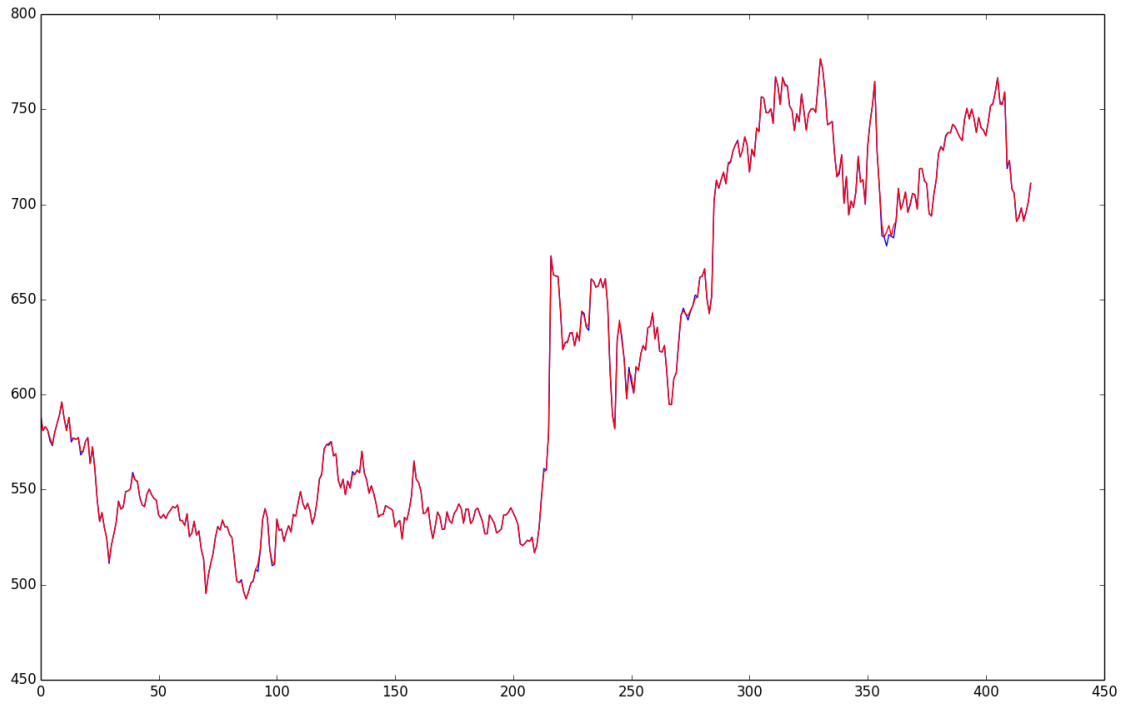
3. Stock splits need to be calculated correctly for accurate reporting of asset.
4. (Revenue) Return since begin is removed from state information for reason discussed in the implementation section.
5. Balance between Exploitation versus Exploration, learning decay rate are tuned.
6. The window (number of days) for rolling average has a big impact on the prediction. This is difference per stock, and dependent on the length of the trading period.

After the tuning mentioned above, the performance of TraderAgent got better (and at least the asset calculation is correct). The tuned TraderAgent produces equivalent or better results than standard "market return" in a lot of cases. Some examples shall be given in the later sections.

Results

Model Evaluation and Validation

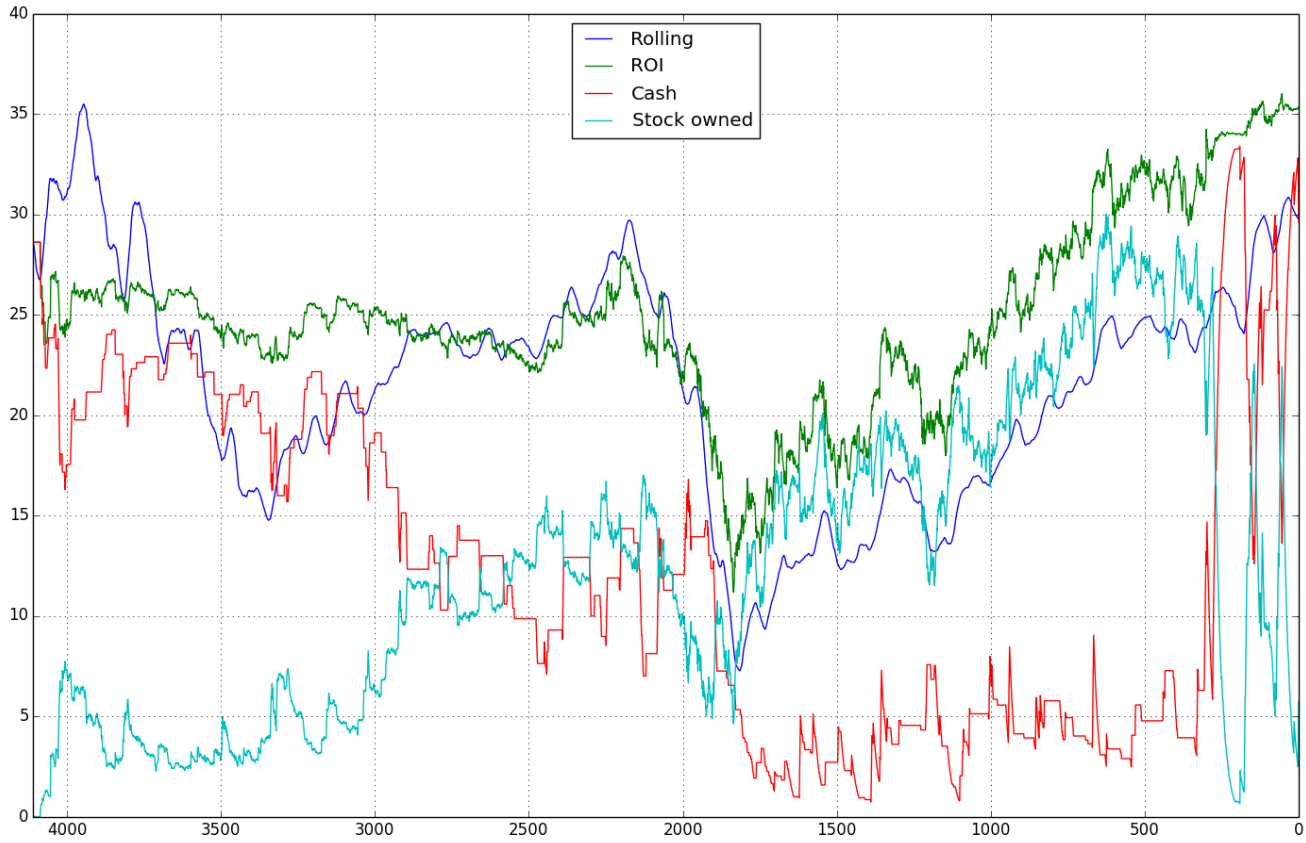
For Gradient Boosting Regression, the evaluation and validation process is straightforward. The predicted result is compared against the target column of the test set. The Mean Squared Error of test set for Tesla stock is for example 0.3423. Mean Squared Error for Google stock is 3.7 (so real difference is 2 dollars, which is around 0.3% of recent Google stock price).
The figure below shows 2 line (red shows prediction, blue is real price) of Google stock. The 2 lines almost overlap which each other. We notice the most visible difference is when there is a sharp "edge" in the real price (big changes in the price). (around X 350)



For Q-learning trader, the tuning and validation process is more extensive:

1. The performance of the algorithm as mentioned is compared against the standard market return.
2. Different number of stocks and time period are evaluated:
    1. “Star” performers like Apple, Microsoft, Google over a long period of time.
    2. “Mediocre” performers like Philips and Bank of America
    3. “Company in crisis” like Yahoo or HP.
    4. Some periods during the dot com burst and great recession.

The figure below shows the performance on Shell stock for the last 15 years:

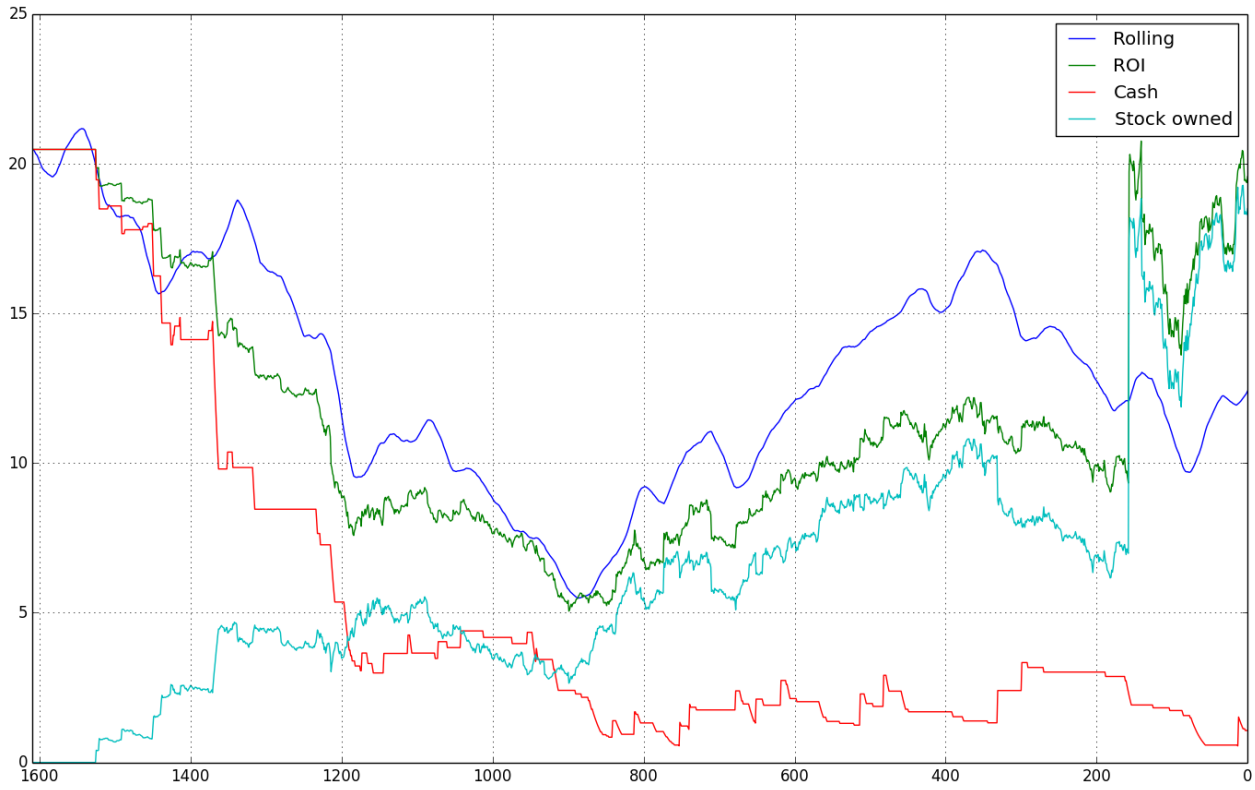


“Rolling” is the rolling average of the Adj Close price.
“ROI” is the return on investment: different between total assets at X axis moment and the begin investment capital.
“Cash” is the total number of cash available at the X axis moment.

“Stock owned” is the total value of stock owned at the X axis moment. Note: big jump in Stock Owned line could represent a “split” in stock.
“ROI”, “Cash” and “Stock Owned” are normalized so the begin point with the Rolling is the same.

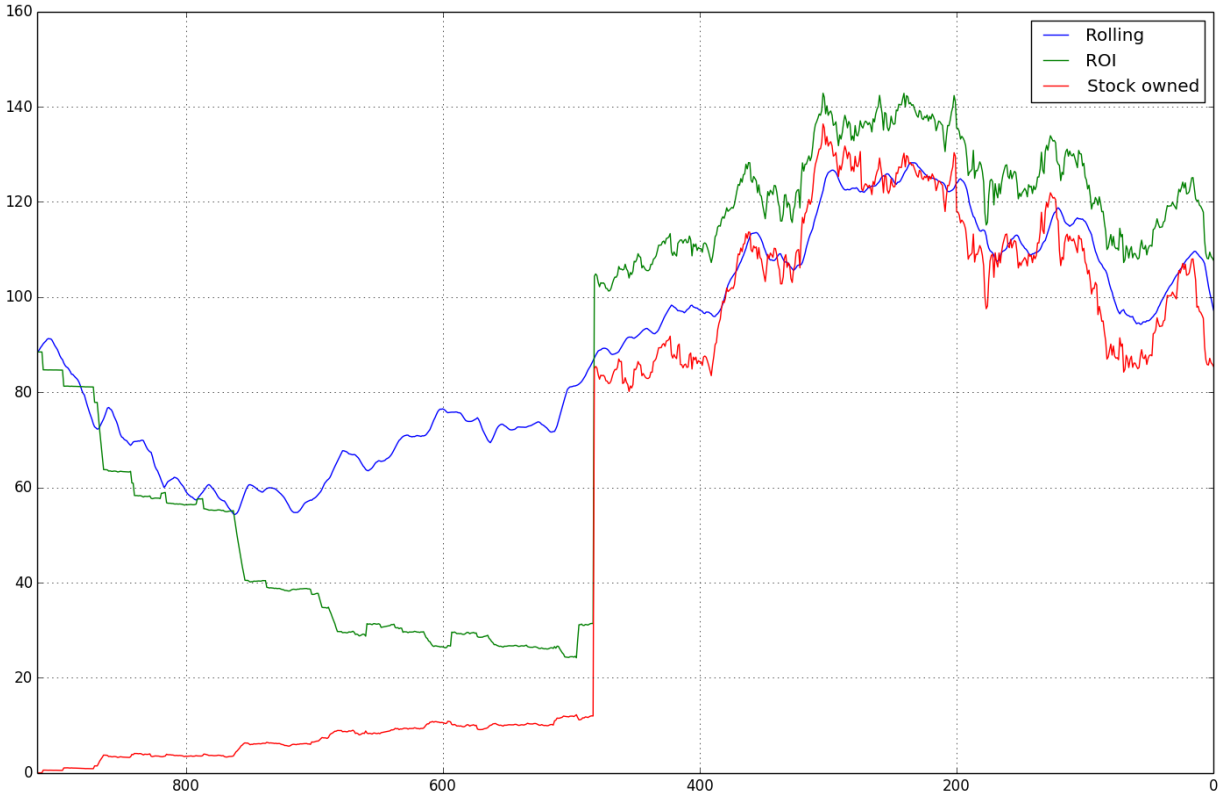
When the ROI line is above the Rolling, it means the Trader Agent is beating the market. When it is lower, it means the Trader Agent is losing to the market.
It is also interesting to notice when the price trend is going down (Rolling down), then our trader agent starts buying stock (“Cash” line down, “Stock owned” line up).

The figure below shows our trader agent trading HP stock, a declining stock (market lost 40% of value, our agent “lost” only 4% of value):



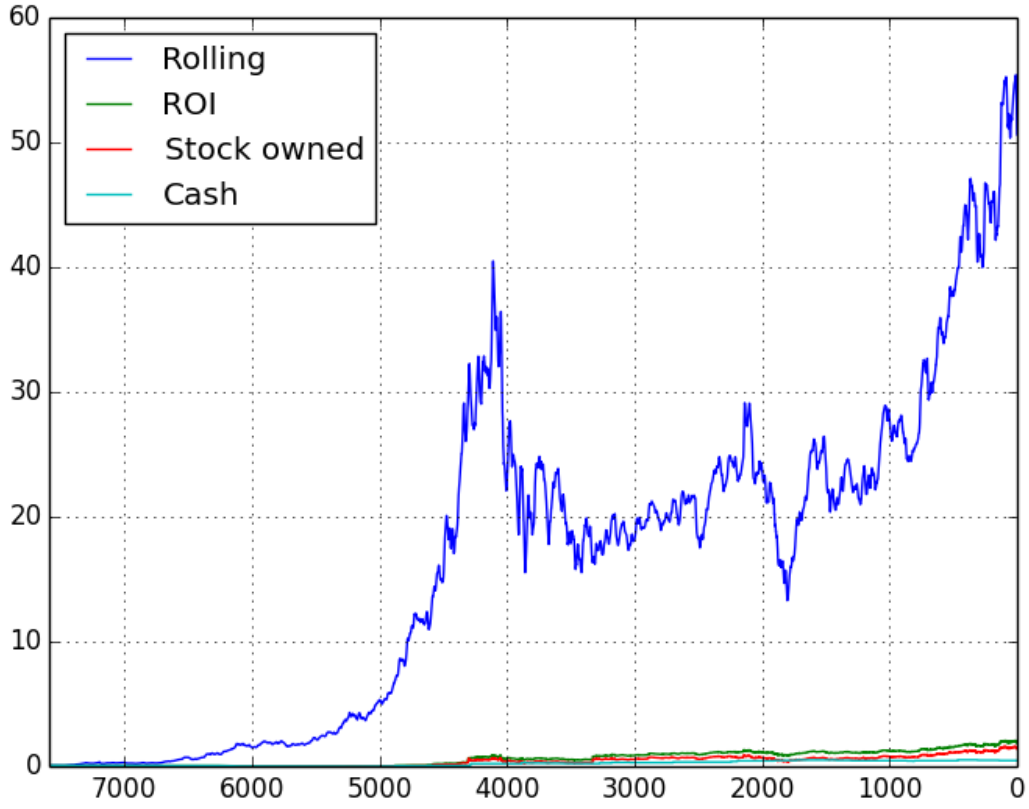
Our learning agent correct predicts a “local minimum” around X point 200 to buy more stocks.

The figure below shows our agent beats the market for Apple stock in the last 4 years (when 21% profit against 10% market price):

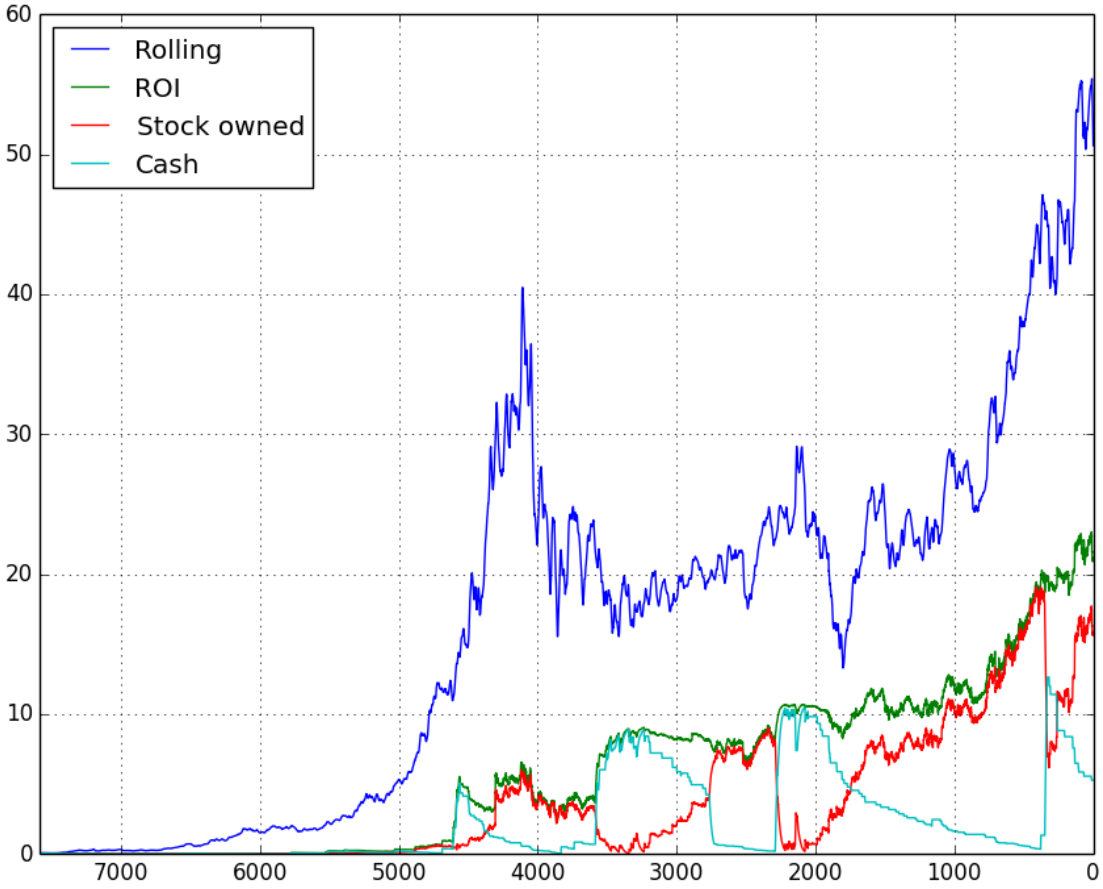


The model correctly predicts a future “jump” in price around point 500 to buy more stocks.

When the default value are not tuned “correctly” (for example with star performers and investment over a long period of time, like Microsoft stocks for 30 years), the agent perform much poor than the market (increase 30-90 times versus 758 of market):



When the default is tuned (e.g. we trust this stock over a long period of time, then we give a higher default for all (state, 'buy'), then the performance is much better (the gain after 40 years jumps to 300 times from 30 times). The only way to approach market situation in this case is simply "buy". In another word, if it is the stock we trust, and the investment duration is long-term, then "buy" should have higher default value than "sell").



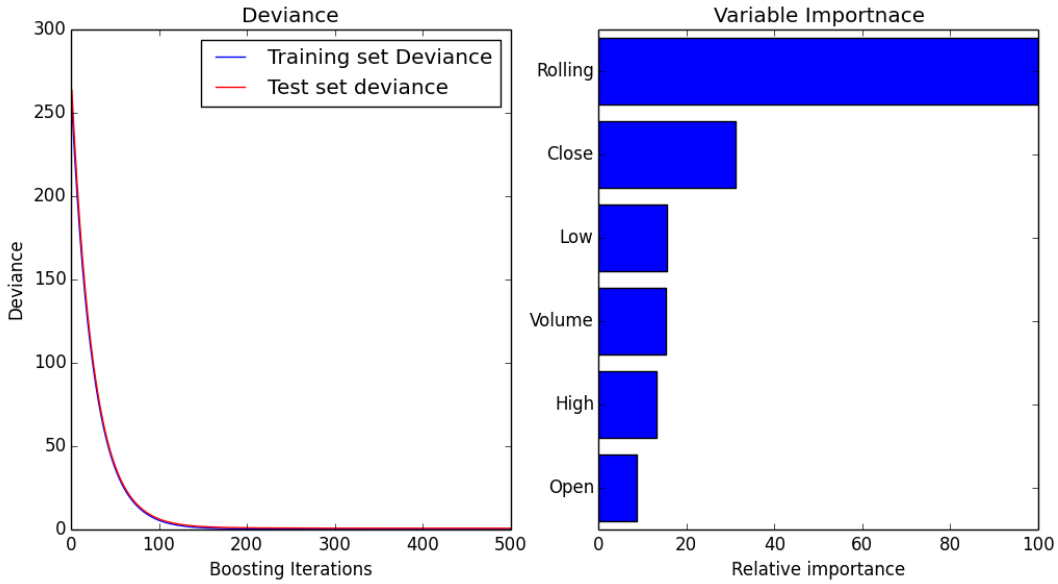
Justification

The result of Gradient Boosting regressor is as good as, or even better than the anticipated result. An extra step should be done to use the regressor model to predict the new stock value of the coming days. However, since the result is so good, there is little doubt here.
The result of the Q learner is discussed above. Beating the market is generally “hard”, if not possible. Like present above, this model perform better than the market in many cases. A lot of improvements can still be done to make this better. Please refer to the improvement section.

Conclusion

Free-Form Visualization

This figure shows the relative feature importance for the Gradient Boosting regressor and the effectiveness of Boost regressor relative to number of iterations (visualization code taken from sklearn website).



As expected, the Rolling average contributes the most to predicting the Adjusted close price, followed by Close price.

Reflection

This project is really fascinating. The thing i like most is the experiment with Q learning to implement a self-regulating trader. What makes it so interesting:

1. You can link what you do to real life “events”.
2. There are so many parameters you can tune that can product different data.

The current state of the program is more an beginning rather than the end. See improvements for future plans.

Improvement

The following can be improved in the Q trading implementation:

1. Create a “trading profile” and optimize the parameters accordingly. For example for long term investment the default should be more “buy” with a large chunk than “sell” or hold.
2. Duration of trading should impact the rolling window in days.
3. Some code clean up & performance optimization. The current state is certainly not product quality code.

On the “application” level, the following can be done:

1. Create an easy interface to tune the TraderAgent.
2. Support trading of multiple stocks (in portfolio).
3. Simple UI.
4. Aggregate data source automatically.

Reference

Course: Machine Learning for Trading @ udacity.com
Skilearn documentation and sample code