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. (for example to download historical price of Yahoo http://finance.yahoo.com/q/hp?s=YHOO)

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).
Stock data provides a wealth of information for machine learning practitioner to train their learning model and make prediction. Predicting what the price of stock will be is also a very exciting, since it can result directly in your wallet.
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 some other derived data (features) to predict Adjusted Close (target)? Adjusted Close is the closing price adjusted for stock split over time. Large public trading companies often have historical stock data for years (e.g. Microsoft for around 40 years), Google 10+ years. For a trader it is often of interest only to predict the price of one day. For our experiment we shall use the stock price of the previous 1 year to train, and predict the price of the last week. For the prediction to have any practical value, we should use the **features of day n** to predict the target of day n+1.
2. Can we train a "Trading agent" to learn from the historical data to it can make independent trading decisions to maximize profit overtime? (please refer to more more elaboration below).

(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 note, 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 Absolute Error (MAE) between predicted Adj Close and actual Adj Close to judge the model performance. MAE is a robust and widely use performance metrics for regression model and is also suitable for stock price prediction. This is also very intuitive to interpret (how much "off" from the price that we predict). The goal is to have this value as small as possible.

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).
(2) is an interesting experiment, we shall use stock prices of different companies in different time periods (e.g. 1 year, 5 years, 10 years) to compare the “machine learning” trader and the human trader.
To illustrate (2) better, let us look at an concrete example.
Say we give a “human” trader and a “machine” trader each 10 000 USD to buy Google stock. The human trader shall just buy all the stocks at once and hold the stock over time. The machine trader shall decide on his own to hold, buy, or sell the stock. The table shows an imaginary price for illustration.

	2016 Jan	2016 Feb	2016 March	2016 April	Return on investment
Stock price	100	90	80	120	
Human, start with 10K	buy 100 stocks	hold, total assets 9K	hold, total assets 8K	hold, total assets 12K	12K/10K = **1.2**
Machine	hold	buy 50 stocks, owns 4500$ in stock + 5500 in cash (total 10K)	buy 50 stocks, owns 80 * (50 + 50) = 8000$ in stocks + 1500 in cash. (total 9.5K)	sell 80 stocks, earned 9600$ + 1500$ cash + 2400$ stocks value remain (20*120) = 13500$	13500 / 10000 = **1.35**

From this illustration, the Machine trader perform worse than human trader in March 2016, but bounces back in April.
We shall evaluate with different type of stocks over multiple begin and end point in times. The goal is for the Machine trader to have **higher** Return on Investment (ROI) **all time.** This is because assume you need the money on any given day, you can sell the stock you earn and still make a handsome ROI without having to wait till a certain deadline.
This can be compared easily by drawing 2 lines, X is time, Y is total assets. At any given time we would want to see the “machine” line higher than the “human” line.

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.

Here is 5 data samples of Facebook stock in recently:

Date	Open	High	Low	Close	Volume	Adjust Close
2012-05-18	42.049999	45.000000	38.000000	38.230000	573576400	38.230000
2012-05-21	36.529999	36.660000	33.000000	34.029999	168192700	34.029999
2012-05-22	32.610001	33.590000	30.940001	31.000000	101786600	31.000000
2012-05-23	31.370001	32.500000	31.360001	32.000000	73600000	32.000000
2012-05-24	32.950001	33.209999	31.770000	33.029999	50237200	33.029999

Date is the day in which the data is observed (something date is not present). Open, High, Low and Close are respectively the Stock Open Price, Highest Price on the stock in that day, Lowest price of stock in that day, Close price of the stock in that day. Adjust Close is the close price of the stock in that day, adjusted for stock split over time. All the stock prices for our dataset are in USD. Volume is the number of stocks traded.

Explain over stock split. Let us have an example:

1. Assume Jan 1st 2016 Facebook close price is at 30 USD per stock (and it has never been split. So both the Close and Adj Close are 30 USD.
2. Assume Jan 2nd 2016 Facebook decide to split the stock buy 2: so for every stock a consumer owns, he/she shall get 2 stocks. Assume the price is "unchanged, relatively". The new close price for the split stock is 15 USD per stock. However, the whole historical data set shall need to be revisited to correct for stock split. The new value for Adj Close of **Jan 1st** shall become 15USD (from 30 USD), and Adj close for Jan 2nd stays 15 USD.

Note: the other values Date, Open, High, Low, Close, Volume are not corrected overtime.

So if we look at the stock price over a long period of time. and for stocks that are split multiple times (say Microsoft), the Close price of April 11 1990 is 41.49, compare to Adjust Close of the same day of over 1.309501.

When making comparison overtime, we shall need to use Adj Close for consistently. Only when making "trade" decision on that day, a trader shall need to "pay" for the price of that day, not the "Adjust Close" price of that day.

Furthermore, 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. As mentioned above Adj Close has correction for stock split.

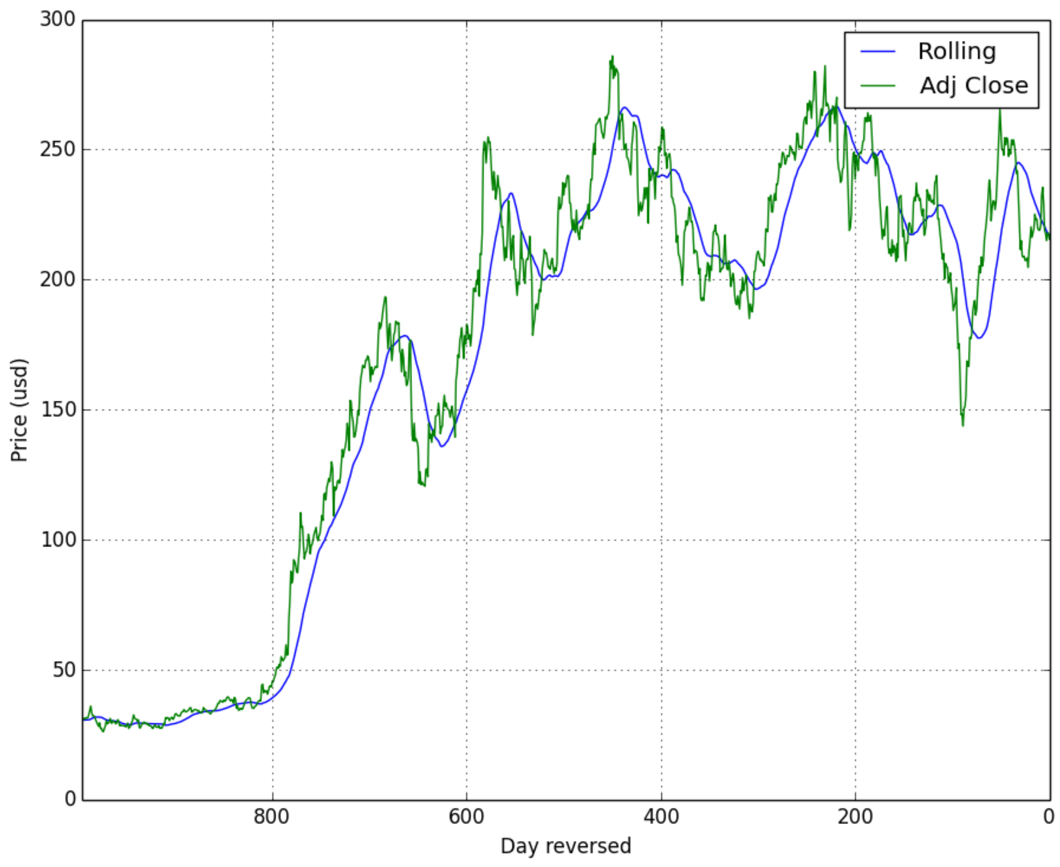
Standard statistical value can be calculated by using DataFrame.describe(). However, for this context, there will not be much value since we analyze a large number of stocks. Also for this assignment the most important data is price over time.

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.

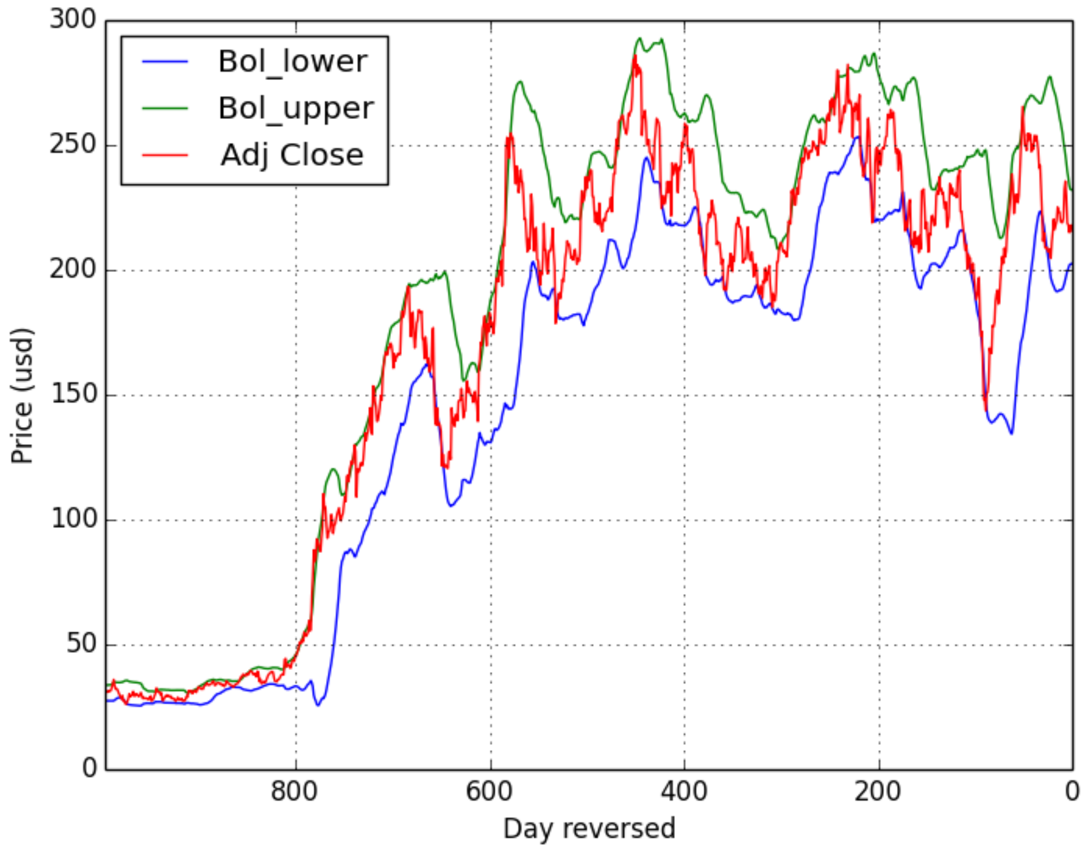
Exploratory visualization

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).



These visualizations meant to show how the correlation of Rolling Mean, Lower/Upper Bollinger bands with the trend of future stock price. For example if the price is lower than the rolling mean, it tends to go up (after some times). And if price is higher than the mean, it tends to go down. The trends of price with regards to Bollinger Bands are explained above.
This is especially important for understand how can we use these information to “teach” our machine trader to make buy, sell, or hold decision (e.g. if the price shall go up, let’s buy).

Algorithms and Techniques

Question (1) - predicting Adjust Close price is a typical regression type problem: from a number of features (e.g. Open, Close, Volume, Rolling Price) we need to predict the target - Adjust Close. The target is a continuous value, therefore it is a regression problem, NOT a classification one.
The following regression techniques are widely use:

1. (Ensemble) Gradient Boosting
2. Support Vector Machine (SVM)
3. Decision tree
4. Bayesian linear.

Since the focus of this project is on hypothesis (2), no extended analysis is provided on comparing and contrast these regression techniques. It is decided to use Gradient Boosting and SVM because of their strengths in linear regression dataset in general (and this dataset is no exception).

The core idea behind gradient boosting is to turn multiple “weak” learners into “strong” learners through iterative steps. Each step of a “weak” learner can be thought as simply using 1 feature column to predict the outcome. A number of “wrong prediction” shall be mixed with the “right prediction” for the next iteration. Multiple iterations shall result in a “strong” learner.
SVM is an “enhanced” linear regression. SVM tries to maximize the margin between the feature values. SVM is recommended because of its high performance for stock data.

The strengths of Ensemble Gradient Boosting which are applicable for our dataset are:

1. Predictive power (produce good results, which is the utmost important factor)
2. Robustness to outliners in output space.

The high predictive power of Gradient Boosting comes with a high performance cost. For the experiment with our “small” dataset (limited number of stocks, stock prices observed in days instead of seconds), this is not a big issue.

Question (2): Training a machine learning agent trader to make the “right” decision is a typical reinforcement learning problem. The core idea is to have the agent (our learning algorithm) interact with the environment (the stock market data). The data of the environment combined with the agent’s state (e.g. stock owned, money owned) are combined into “environment state”. Base on the rewards (win, lost), the agent shall learn and improve and make better decision overtime that will maximize her Return on investment.
There are 3 main reinforcement learning approaches: policy search, value-function based, and model based. One of the most efficient and easy to implement reinforcement learning algorithm is Q-learning. Q-learning is a value function (model free) reinforcement learning technique. Q-learning is efficient because when the states are defined correctly, the algorithm often converges quickly, which means our agent learns faster.

We decide to use Q-learning for question (2) for the following reasons:

1. As mentioned, Q-learning is efficient, easy to implement, and is one of the most popular reinforcement learning algorithms out there.
2. For the stock data set available, the state, action, and rewards can be defined in a way that makes our algorithm efficient.
    1. State space is reasonable defined for fast convergence (see implementation for more details). From the Exploratory visualization section, we see “Rolling mean” and “Bollinger bands” can be used in the state to give our agent “hint” to build a better value table (Q table).
    2. Action of the agent is straight forward (sell, buy, or hold).
    3. Reward is straight forward (profit or loss).

For more information on the detailed implementation of the algorithms, please refer to the Implementation section.

Benchmark

As discussed earlier:

1. For regressor Adj Close predictor: the performance benchmark is obtained by comparing the Mean Absolute Error between predicted Adj Close and the real Adj Close of the test data set for the test set (1 week/ 5 working days). **The desirable result is the average Absolute of the predicted period should be less than 3% average Adjust Close price of the stock in that week (test duration)**. E.g. say the stock price for the week is 10, 11, 12, 13, 15. The predicted price is 10, 11, 12, 13, 14. Then the average MAE is 1/5 = 0.2. The average real stock price is 12. The MAE is thus ~ 0.2/12 = 1.67% of the stock price, which meets our goal.

2. For Q-learner trading agent: the performance benchmark is the difference of Return on Investment of the trader agent (ROI agent) and the return on investment of the human trader (buy at the beginning once all, hold always) (ROI human). The difference is done at the end of the trading period. The goal is ROI agent > ROI human for at least 50% of the observed stocks and observed periods. Please refer to “Metrics” section for calculation of the ROI.

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). Furthermore, 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).

As briefly mentioned in the introduction, for the regression solution, we shall shift the target column 1 day to the left our algorithm has more practical value: we would like to use the input data of day n to predict Adjust close of day n+1. We shall then add the “shifted” Adjust Close to the list of “feature” columns as well (this is no cheating since the Adjusted Close is from yesterday, NOT today, which is perfectly “legal”).

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.

Note: when multiple stocks are used, it is a good scale to scale the stocks price to the same threshold (e.g. say Google stocks at 600 should be scaled down to make it comparable to Microsoft stock at around 40).

Implementation

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 test set to prevent model over fitting and use test set to validate the result. Note: splitting the stock data into train and test set should be time dependent: train set should be "earlier in time" (say first 265 trading days) and test set should be "later in time" (say, last 5 trading days).
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, compare to performance benchmark, 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 with different dataset using the performance benchmark defined earlier.

The code 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

Regression refinement

The following refinements have been performed for the stock regression:

1. “Enhanced” the data set with “rolling average data” to make better prediction.
2. Use GridSearch for GradientBoosting regressor to find the optimize set of parameters.

Please refer to the Results section for comparison between “untuned” algorithms and “tuned” algorithms. For brevity, only the difference in results of step (2) (use GridSearch) is mentioned.

Trader agent refinement

For the Q-learning trader implementation, the parameters refinement process more 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. If the Q table is properly filled, than the use of more “Exploitation” shall help the trader stay closer to the mean (less loss in case stock price decrease, less gain in case stock price increases).
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.
Please refer to the “Results” section for some “Before” and “After” examples.

Results

Model Evaluation and Validation

Regression results

For Gradient Boosting Regression, the evaluation and validation process is straightforward. We use the metrics and performance benchmark mentioned earlier in this document. As a reminder, the training period is 1 year, the test period is the last 1 week. The performance metrics is ratio of Mean Absolute Error / Average real Adjust close price for the test period. The goal (benchmark) is to have the result < 2% (**the lower the better**).

The table below compares prices of Google, Facebook, and Tesla stocks with both Gradient Boosting and SVM **(all using default values)**

	Google	Tesla	Facebook
Gradient Boosting Test Set	**1.0633%**	**2.4583%**	**1.2414%**

SVM test set	**0.4258%**	**5.1782%**	**9.1270%**
Gradient Boosting train set	0.5418%	1.0322%	0.5566%
SVM train set	8.9662%	9.2683%	6.2149%

From the results above we can draw the following conclusions:

1. Gradient Boosting yields more stable (predictable) results in the 3 datasets.
2. Results of Gradient Boosting is in the "**acceptable**" range of the original proposed benchmark (3%).
3. Gradient Boosting "overfits" (relatively) more compared to SVM.

Since Gradient Boosting yields better results for our dataset, it is chosen to be optimized further with GridSearch. The following table shows the results of Gradient Boosting before and after GridSearch for our dataset. (Note: this is a different run compared to previous table, so the result of Gradient Boosting default params can differ slightly)

	Google	Tesla	Facebook
Gradient Boosting default params	**1.0820%**	**2.4656%**	**1.2414%**
Gradient Boosting "Boosted" (GridSearch)	**0.4514%**	**2.1376%**	**1.3188%**

It can be observed that the GridSearch improve the results by factor 2 with Google dataset, ~10% with Tesla data set.
For the Facebook dataset: an interesting thing happens: **the "optimized" Boost performs "worse" than the default boost for the test data set**. This is due to GradientBoosting overfit the train set. (0.3681 for optimized version compared to 0.5566 compared to the unoptimized version).

Trader agent results

The following table shows the comparison of our trader agent with the standard "market" trader for a different number of stocks. The performance metrics is ratio of return on investment at the end of the trading period compared to at the beginning. Refer to "Performance Metrics" section for a calculation example of the performance metrics. The trading period is chosen to be 2 years (to give our machine learner some slacks). A number of run through the data is performed (to give the Q table amble data to converge), only the last run is reported here. (Note: < 1 means making loss, > 1 means making profit)

	Market gain	Machine gain (trader agent)
Walmart	**0.947**	**0.978**
HP	**0.889**	**0.938**
Yahoo	**1.063**	**0.927**
Philips	**0.881**	**0.992**

Shell	**1.207**	**1.062**
Google	**1.313**	**1.167**
Facebook	**1.949**	**1.19**

The following observations can be drawn from the data above:

1. Our trader agent generally performs better than the stocks which are in “decline”: Walmart, HP, Philips.
2. Our trader agent performs less good for stocks that have good return on investment: Yahoo, Shell, Google, Facebook. The default parameter is apparently not “aggressive” enough.

Our trader agents tend to stay “closer to the mean”, which is 1. This is understandable, since we use rolling mean and Bollinger bands for the state, which explains the staying closer to the mean tendency. Also generally it is very hard to beat a “star” stocks (simply going up in the long run). A simple strategy is just to buy at the beginning then hold (The Warren Buffett strategy).

Q-learning tuning results

When we tune the trader agent to explore more (use less of the existing learnt information, but choose more for randomness) from max 3% to 15% (of the choices), we get the following results:

	Market gain (same as above)	Machine gain (3% explore)	Machine gain (15% explore)
Walmart	**0.947**	**0.978**	**0.983**
HP	**0.889**	**0.938**	**0.922**
Yahoo	**1.063**	**0.927**	**0.936**
Philips	**0.881**	**0.992**	**1.0006**
Shell	**1.207**	**1.062**	**1.113**
Google	**1.313**	**1.167**	**1.145**
Facebook	**1.949**	**1.19**	**1.289**

This results show a slight improvement compared to the previous “default” results. Biggest improvement can be seen with the “star stock” Facebook (from 1.19 to 1.289).

When the know the marketing is going “UP” (e.g. recession is just over), we can decide to have higher default Q value for **buy** and **hold**. The table below shows the results when default for Buy is changed from 10 to 1000, and hold is changed from 0 to 500. (Max explore rate stays at 15%).

	Market gain (same as above)	Machine gain (3% explore)	Machine gain (15% explore, higher default Buy and Hold)
Walmart	**0.947**	**0.978**	**0.964**
HP	**0.889**	**0.938**	**0.925**
Yahoo	**1.063**	**0.927**	**0.983**
Philips	**0.881**	**0.992**	**0.90**
Shell	**1.207**	**1.062**	**1.07**
Google	**1.313**	**1.167**	**1.22**
Facebook	**1.949**	**1.19**	**1.34**

We see clearly that an improvement is shown with the "UP fast" stocks like Google and Facebook. This makes the case for creating different parameters profile for different stock groups (more trust/ less trust), market timing (e.g. just after recession or recession might be coming because of Brexit), and investor profiles (e.g. more risk with more exploration, less risk then with more exploitation).

Justification

From the results discussed above we observe:

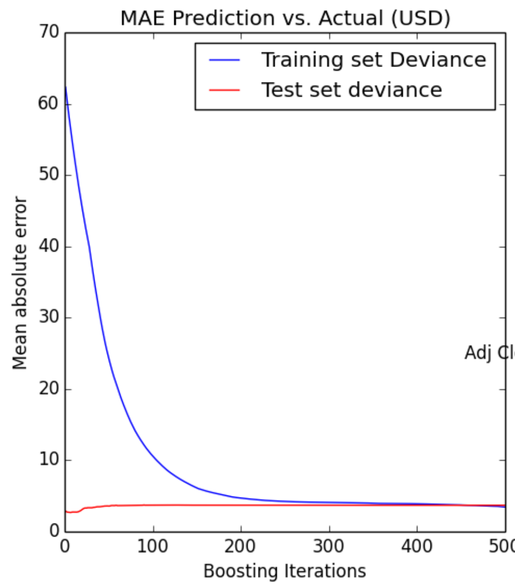
1. The regressor performs well with predicting stocks price for a short period (5 days). The longer period results are experimented, but not shown here. Predicting stocks a long term in advance has little practical values nor accurate.
2. The regressor exceeded the expected performance benchmark (3%) using own predefined metrics (Mean Absolute Value) using our dataset.
3. Using the ratio of Return on Investment metric, the Q learner performs
    1. Better than the market when the stocks go down.
    2. Less good than the market when the market tends to go up a lot.
    3. Different default values/ parameters impact the performance. Tuning the correct parameters require some market/ company information (e.g. technical analysis of company performance can be used to get better default parameters).

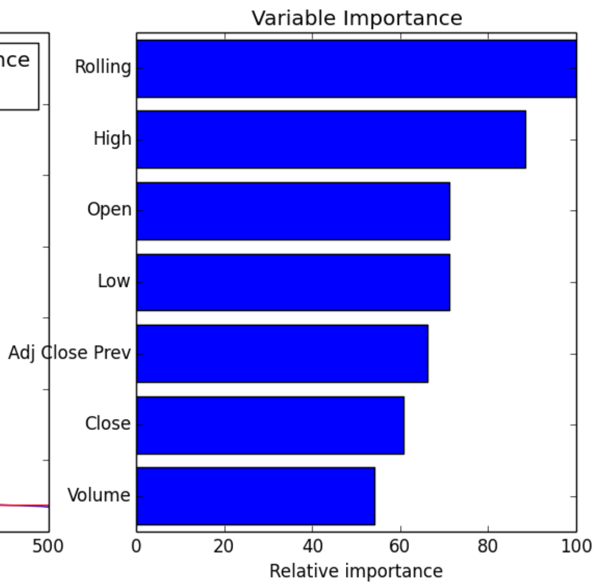
4. It shall be fairer when judge Q learner performance using a mixture of stocks portfolio. We shall leave this for future improvement.

Conclusion

Free-Form Visualization

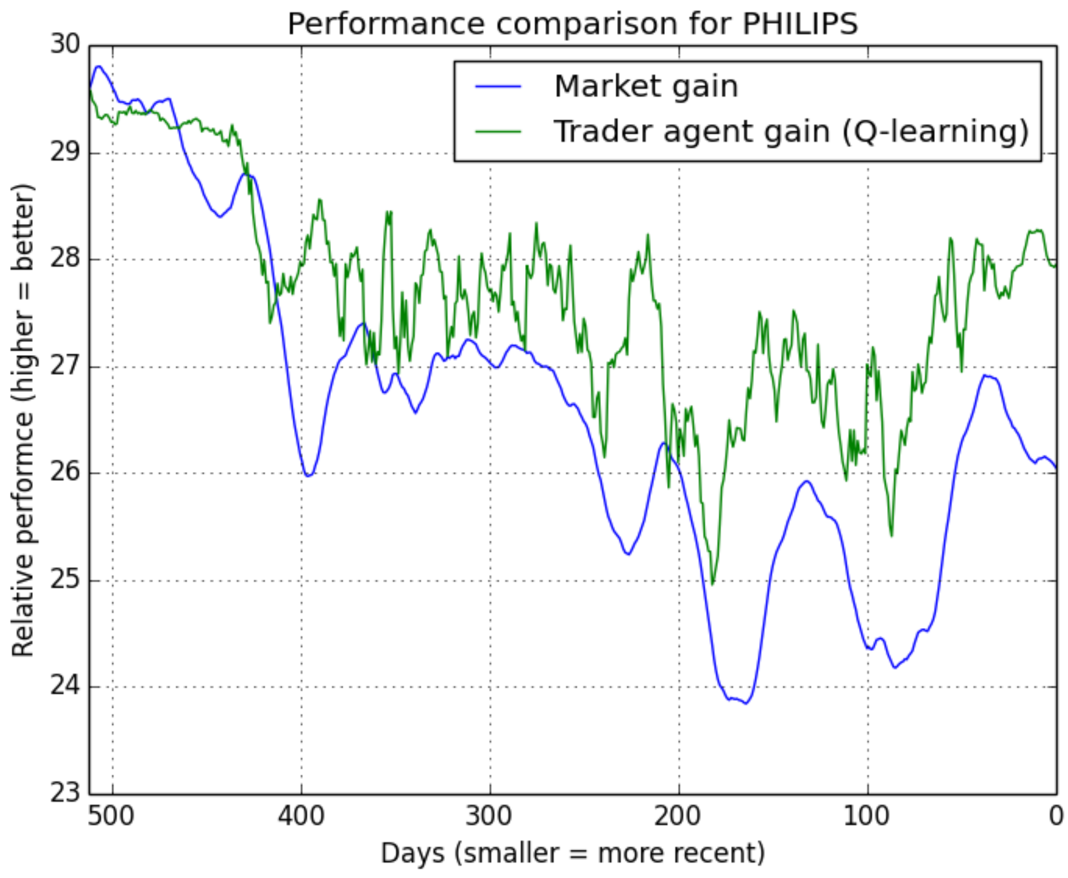
This figure shows the relative feature importance for the Gradient Boosting regressor and the effectiveness of Boost regressor relative to number of iterations. Google dataset is used for this case. (Note: the features are of previous day).



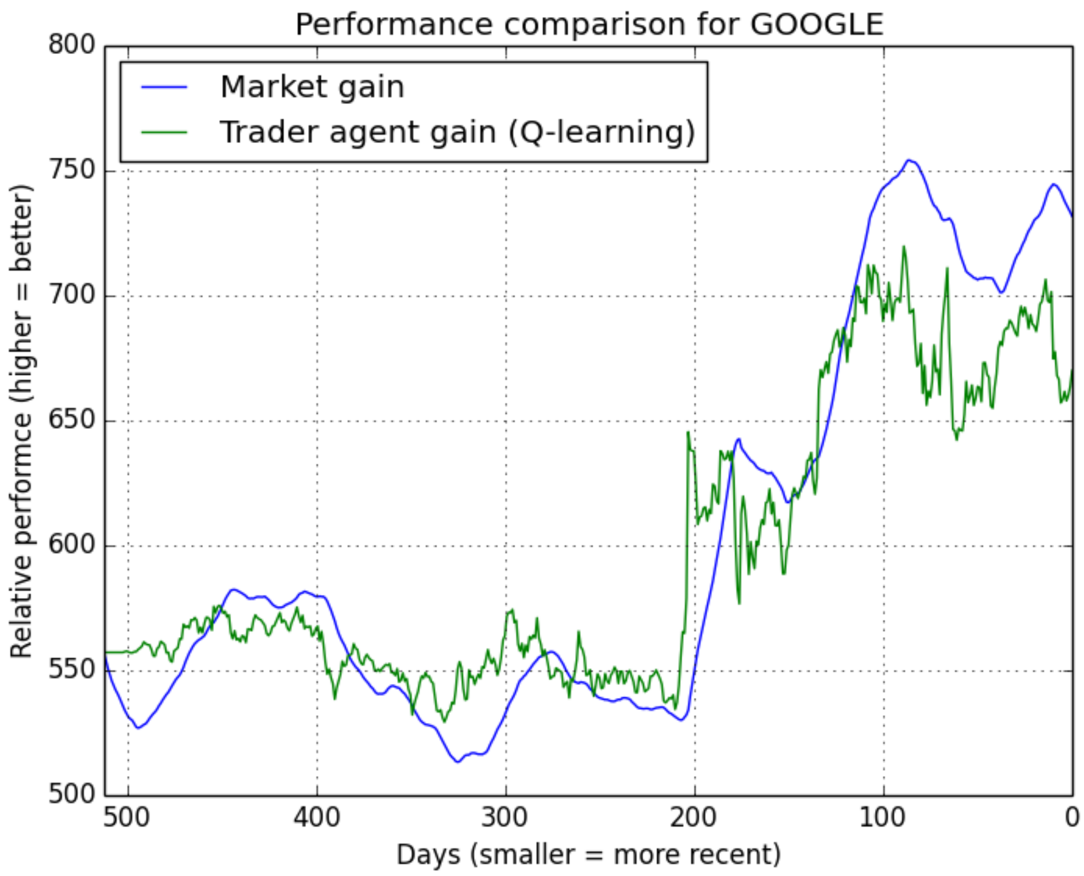


We also observe that the mean absolute error converge after around 200 iterations.
As expected, the Rolling average contributes the most to predicting the Adjusted close price.

Below are a number of comparisons of gain of market compared to trader agent for a number of stocks.
Note: the y axis show the value of the stock. The trader agent gain is the total assets, normalized by this begin value for a relative comparison.

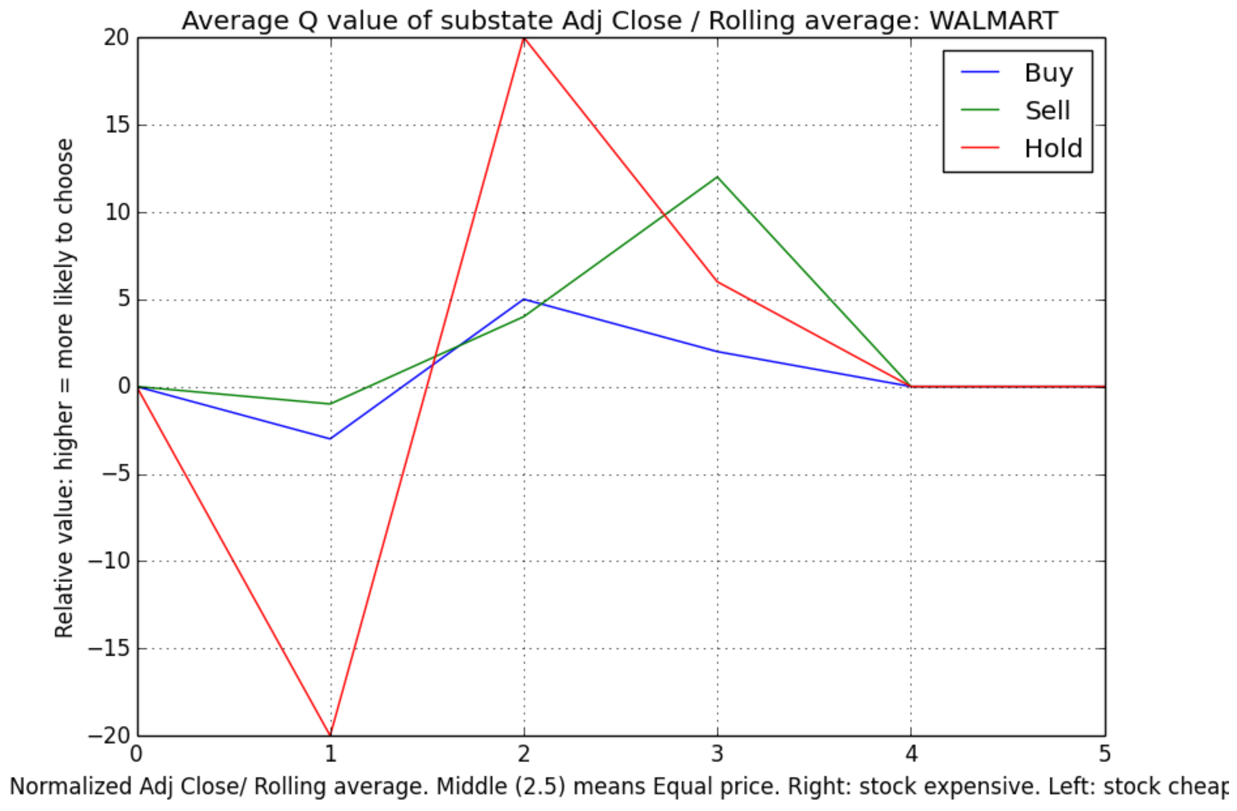


We see for Philips stock, the trader agent performs better than the market for most of the duration (not only at the end). So theoretical the agent can “cash out” anytime and still win the market.



The trader agent perform almost as good as Google price for most of the period, but it lose steam when Google price increases aggressively between days 100 and 150.

The figure below shows the value of a sub state of Q-table (ratio of Adjust Close price over rolling mean: lower means the stock is relatively cheap historically, higher means stock is relatively expensive). The values of other stub states (Bollinger bands) are averaged for this.



We see that when the stock becomes more expensive (>2.5 in x axis), the Q-learner learns it's the moment to sell (sell value higher than buy value) because the stock is more likely to bounce back to the mean). When the stock is relatively cheap (at 2), the buy value get higher (because the expectation is stock price is more likely to increase).

We normalize our substate space to between 0.5 and 1.5 (which is a relatively large range). Typically for a duration of 2 years, the stock change range is between 0.8 an 1.2. This can also be an improvement for the next version. We also observe the more stock data and longer period we use to train the Q-learner, the wider range (e.g. 0,1 and 4,5) we can get with the Q table.

Reflection

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

1. Making prediction is fun. Making prediction about money is even more fun.
2. You can link what you do to real life “events” that has to do with money.
3. There are so many parameters you can tune that can product different data.
4. The goal of “beating” the market is closed to a “mission impossible” goal with the current nativity of the dataset, parameter tuning process, and implementation. However, it is really fun to see the trader agent can really “beat” the human trader a number of times.

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

The steps employed during this assignment are:

1. Get informed about different aspects of machine learning for trading by doing Udacity course “Machine Learning for trading”. Understand where can we get the dataset, what are the main characteristics of the dataset, what can we derived from the data set to make better prediction (e.g. rolling mean, Bollinger band…). Understand the application of machine learning algorithms with stock data set.
2. Download dataset and play with them.
3. Think about what we can do with the dataset. What are the interesting questions and predictions we want to do with the dataset. The regression implementation first came to mind (as was also suggested in the assignment).
4. Implement and improve the regression and Q learning algorithm.
5. Create the report.
6. Rework the review feedback: for all the projects at Udacity, this is the step where students learnt the most. Most, if not all of the feedback are excellent and help push the students to achieve better results and learn more. For example after the first feedback round i found there is a major flaws with the way regression is done with stock data (all “feature data” should be shifted by 1 day to make it having the right “practical” values).

Looking back i was really consumed with improving the algorithm without collecting enough data along the way to show the improvement step by step. This can really be improved during the process.

Improvement

The following can be improved in the Q trading implementation:

1. Create a “trading profile” and optimize the parameters accordingly. A couple of examples for these profiles are “Blue chip, long term”, “Start up, short term”, “Star start up, long term”.
    1. For “long term” investment we would prefer to have longer Rolling average window. For “short term” investment we would prefer to have shorter Rolling average window.
    2. Calculate and update volatility of the stocks to automatically tune some of the parameters. For example, for more stable stock (e.g. Blue chip): a longer rolling average window, higher default Q-value for Buy and Hold. For more volatile stock, a shorter rolling average window, higher default Q-value for Sell.

    3. For the “Star / blue chip/ high performers and long term”, we would like to follow the Warren Buffet style, which is similar to our human trader of “buy at the beginning with all the money you have, then keep the stocks for years to come”. This strategy is particularly effective for stocks like Microsoft, Apple, Google.
    4. Technical analysis (calculate Price / Earning ratio, historical performance) can be used to tune parameters correctly.
2. Implement a policy/rule based algorithm and compare to the performance of Q-learning. A couple of policy can be particularly effective:
    1. If the Adjust close price is higher than rolling mean, then sell. And vice versa.
    2. The same can be applied to Bollinger Band.
    3. Some of the “rules” mentioned above (regarding stock profile).
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