Definition

Project Overview

The purpose of this project is to implement a program, Auto Trader, that attempts to learn to trade stocks like an experienced stock broker. As opposed to creating a model to predict prices and then acting on the predictions, the program is trained to take actions based on changes in the stock prices relative to other computed statistics such as moving averages. The goal is to create an "automated" trader that can outperform a basic "buy and hold" strategy in different market conditions – maximizing gain in up markets and minimizing loss in down markets.

Problem Statement

Predicting stock prices would be a great win for machine learning in the financial sector as finding a reliable solution would allow investors to reap great profits from stock trading. Unfortunately, the market proves to be difficult to predict from past data and some economists claim behaves as a "random walk" than cannot be predicted at all[1]. While some have achieved limited success predicting prices using Neural Networks[2], Support Vector Machines[3], and Hidden Markov Models[4], the predictions have a limited useful time span (days) and aren't reliable enough to automate a trading process.

While predicting actual prices may be difficult, there are trading strategies that are effective in using stock trends to determine when to buy and sell stocks[5]. Specifically, by using moving averages, and stock price changes relative to those averages, the trader can determine when a stock is likely to trend upward to trend downward and make buy/sell decisions accordingly. Figure 1 illustrates this point using Apple stock data for 2016. You can see that when the stock price crosses above the moving average, it is likely to stay above the average for a period of time, and, when it moves below, it is likely to stay below. The points where the stock price moves above or below the average are called "cross-over" points and it is at these times when the trader decides to buy or sell stock.

[1] Wikipedia – Random Walk Hypothesis, https://en.wikipedia.org/wiki/Random_walk_hypothesis
[2] J.H. Choi, M.K. Lee, and M.W. Rhee. "Trading s&p500 stock index futures using a neural network". Proc of the Third Annual International Conference on Artificial Intelligence Applications on Wall Street, pages 63–72, 1995.
[3] L. Cao and F.E.H. Tay. "Financial forecasting using support vector machines". Neural Computation and Application, pages 184–192, 2007.
[4] "Stock Market Prediction Using Hidden Markov Models", Aditya Gupta, Non-Student Member, IEEE and Bhuwan Dhingra, Non-Student member, IEEE.
[5] "Moving Averages", Investopedia, http://www.investopedia.com/university/movingaverage/

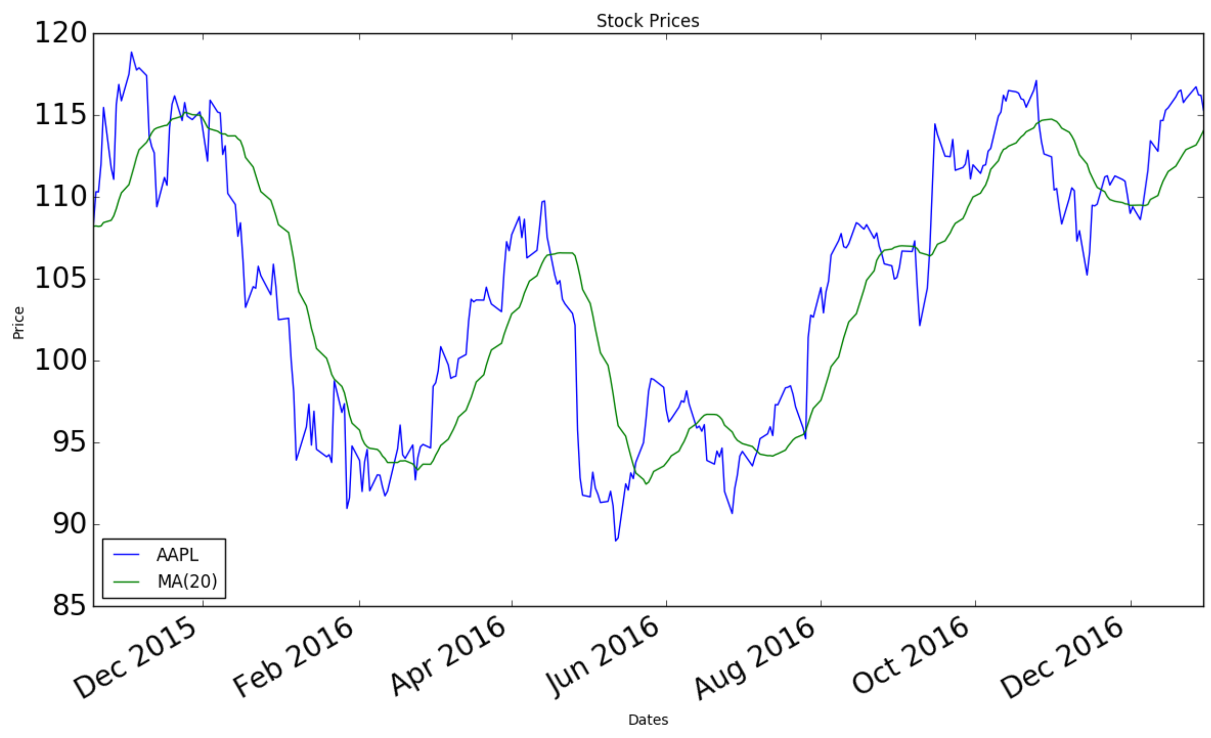


Figure 1

Complex trading strategies can be using moving averages of different time periods (e.g. 10 days, 100 days, 200 days) relative to the daily stock price and other technical indicators such as the Moving Average Convergence Divergence (MACD) which compares two moving averages of different time windows, trading at the cross-over points between the two averages[6].

The goal of this project is to build an automated trading platform, Auto Trader, that can not only learn these known strategies but also potentially uncover new ones. Using a reinforced learning approach, Auto Trader is trained on historical stock data and derived technical indicators such as moving averages to buy and sell stocks with the goal of maximizing the investment value over time. During the learning process, the automated trader is rewarded if it makes a good decision (e.g. buys a stock that then increases in value) and penalized if it makes bad one (e.g. sells a stock that subsequently increases). Multiple stocks will be used in the training process over a lengthy time period (e.g. one year) so as to generalize the training results.

Once trained, Auto Trader will be used to simulate trading over a specified testing time period using historical stock data that is beyond the range of the training set. The simulation starts with a portfolio of stocks and an initial investment for each stock. For each day in the testing period, Auto Trader is then provided the daily stock price and derived indicators for each stock in the portfolio asked to determine if it should buy, hold or sell the stock. At the end of the test period, the simulation calculates the net gain (or loss) in value of each stock in the portfolio based on the buy/sell decision made. This outcome is then compared to the value that would have been obtained by simply buying the stock at the beginning

[6] "Moving Average Convergence Divergence.", Investopedia, http://www.investopedia.com/terms/m/macd.asp

of the test period and selling it at the end. The expectation is that Auto Trader can do better than this "buy-hold" approach.

Metrics

As stated, the Auto Trader is measured on the net gain(loss) it achieves for a portfolio of stocks while trading over a given time period. When a stock simulation starts, Auto Trader is given a specified amount of cash (e.g. $10,000) for each stock in the portfolio. When the simulation ends, Auto Trader will have a combination of cash and stock in the portfolio. The difference between the portfolio value at the end of the simulation and at the beginning of the simulation is the net gain/loss. This is expressed in formulas as show below:

$$initial\ value = number\ of\ stocks \times initial\ stock\ investment$$

$$final\ value = \sum_{i=1}^{number\ of\ stocks} cash(n) + shares(n) \times price(n)$$

$$net\ gain(loss) = final\ value - initial\ value$$

Analysis

Data Exploration

Auto Trader uses historical stock price data from Yahoo! Finance. Data elements returned from the API include, Date, Adjusted Close, Close, High, Low and Volume for a specified stock and date range. Open, Close, High and Low are the price at the opening of the trading day, price at end of the trading day, highest price reached during the day, and lowest priced reached during the day. These prices are the actual prices that were recorded on that trading day. Adjusted Close is the closing stock price adjusted to account for stock splits, dividend payouts, and other non-market stock price changes so that the stock can be accurately compared day to day over time[7]. For example, if a stock was trading at $100 one day and then underwent a 2:1 stock split the next day so that it was trading at $50, we would not want to see this as a 50% decline in the value of the stock. Since we are dealing with historical comparisons of stock prices in this project, Adjusted Close will be used.

Data is only returned from the API for days in which the market is open which excludes weekends and certain holidays. There may also be the case where the market is open but trading on a specific stock is not occurring (rare). In this case, nothing will be returned for that stock for that date. These potential time gaps will have to be considered when comparing multiple stocks.

The bulk of the data used in this project is derived from the adjusted closing price. It includes moving averages of various time windows, Bollinger Bands which define and upper and lower boundary two standard deviations from the mean, and daily returns which compare one day's price to the prior day's price. The details of how these are computed will be discussed later in the pre-preprocessing section.

[7] "Adjusted Closing Price", Investopedia, http://www.investopedia.com/terms/a/adjusted_closing_price.asp

Exploratory Visualization

As you can see below comparing stock prices for Apple, AT*T and the S&P 500 Index (all stocks normalized for comparison), daily stock prices are very noisy with no clearly repeatable pattern. There are times when the stock is trending upwards such as AT&T between January and June 2016, relatively flat such as Apple between April and June 2015, and trending down such as Apple between November and January 2016.

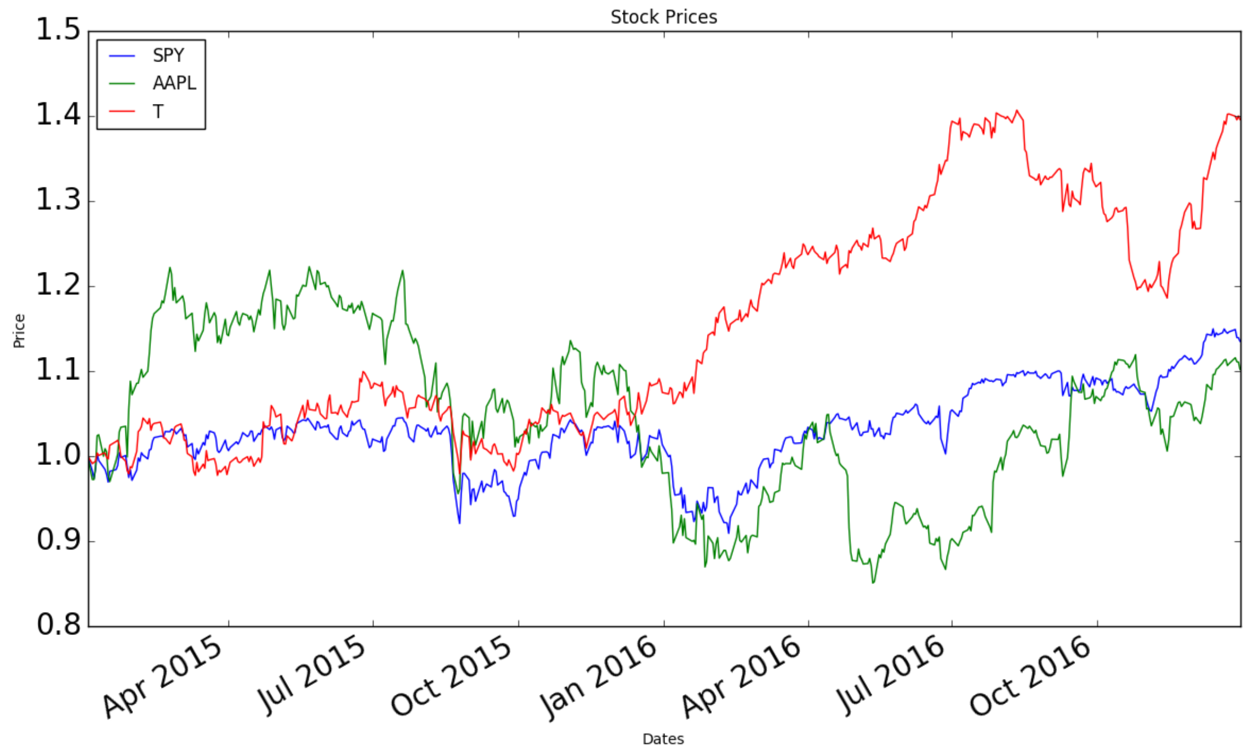


Figure 2

As stated before, trying to predict the stock price from stock data has proven to be difficult. The chart below shows a Support Vector Regression Model fitted to historical data and the prediction it makes outside of the training period[8] . While this is by no means a finely tuned model, it highlights the difficulty in making long term predictions for stock prices.

[8] See the Jupyter Notebook associated with the project for results from other models as well.

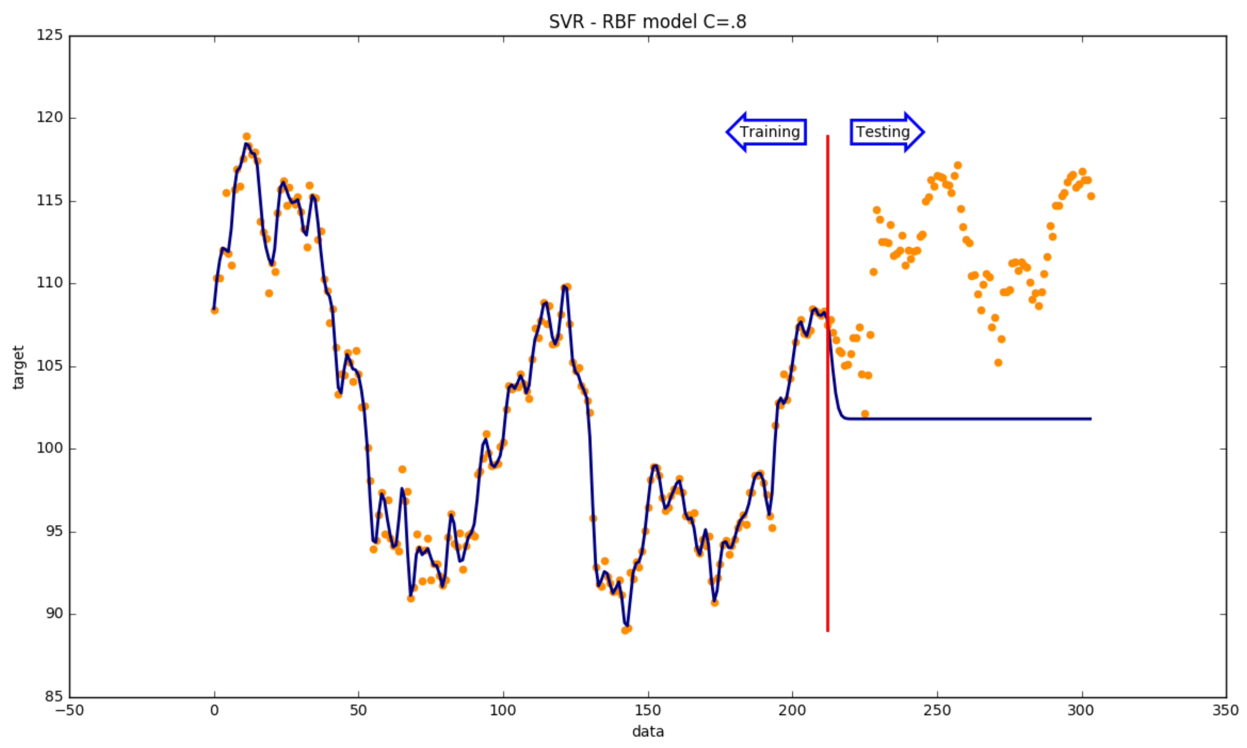


Figure 3

Moving averages tend to smooth out the noise in the stock data. The longer the time windows used for the average the smoother resulting curve as shown below.

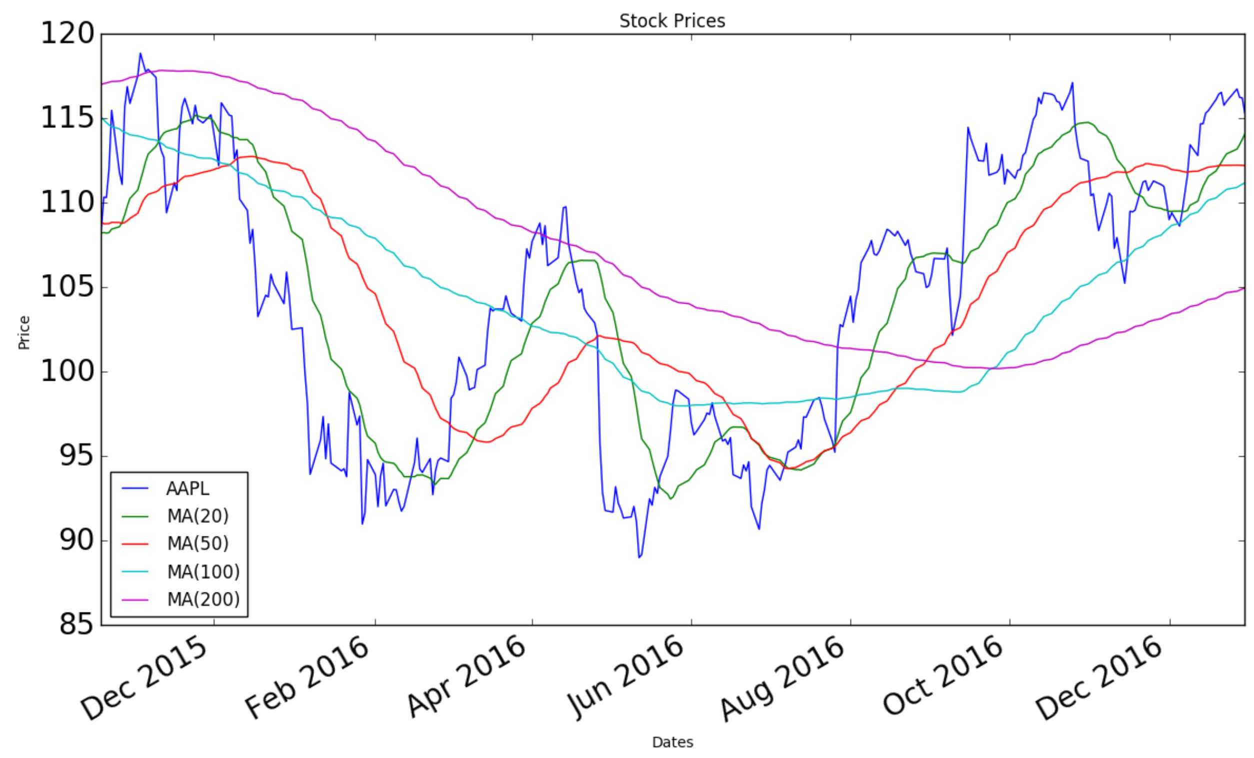
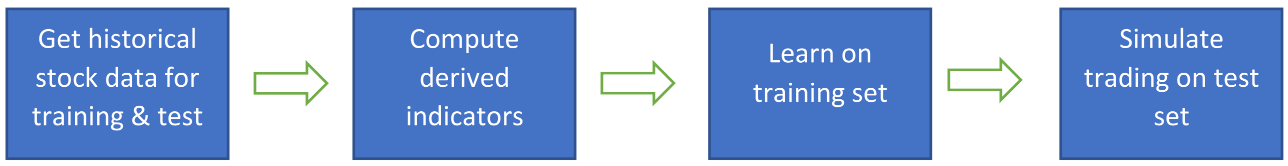


Figure 4

This graph visually reaffirms the assertion made earlier that stocks tend to stay above a moving average once they cross above and stay below once the cross below. The longer the time window used for the moving average, the more likely this is to be the case. The figure also highlights, however, that these assertions are not "fool proof". Note that in November 2015 and April 2016 Apple stock peaked above the 200 day moving average only to immediately fall back below.

Algorithms and Techniques

The general processing flow for Auto Trader is as follows:


Get historical stock data for training & test
Compute derived indicators
Learn on training set
Simulate trading on test set

Basic Parameters and Setup

The basic parameters for running Auto Trader are a list of stock symbols (the portfolio), an initial dollar amount to invest for each stock, a date range for training, and a date range for testing. The user can further specify which features or indicators to use in the training and test simulation (see below for an available list).

Loading Data

Auto Trader reads historical adjusted close stock prices for the designated stock portfolio for the specified training and test time periods. As stated before, adjusted close is used so that day to day comparisons can be made for a specific stock. To allow for the computation of 200 day moving averages, Auto Trader will start the load 200 days earlier than the start of the training period. The training and test periods are assumed to be contiguous with the test period later than the training period although this is not necessary to achieve the desired results.

Computing Derived Indicators

Once historical prices are load, Auto Trader computes technical indicators or features. These include Moving Averages, Bollinger Bands and Daily Returns.

The Moving Average is the simple mean for the a given time window (n = number of days in window, t = current day).

$$\frac{1}{n}\sum_{i=0}^{n-1} price(t-i)$$

For a 20 day window, the moving average is the mean adjusted close price for the current day and the 19 days prior. I used a simple unweighted mean here although some traders use exponential [9] weighted averages which give more emphasis to recent prices (something to be explored in the future).

Bollinger Bands are upper and lower limits 2 standard deviations (SD) away from the mean (MA). The bands for window n are computed as

$$Upper\ Bollinger\ Band\ = MA(n) + 2 \times SD(n)$$

$$Lower\ Bollinger\ Band\ = MA(n) - 2 \times SD(n)$$

Daily returns are computed as

$$(current\ day\ \div prior\ day)\ - 1$$

Daily returns are computed for the Adjust Close Prices and the Moving Averages.

Averages and bands are computed for 20, 50, 100 and 200 day windows. The following table details the features/indicators used:

Indicator	Label
20 day moving average	MA(20)
20 day moving average return	MA(20)_return

[9] http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:moving_averages

Indicator	Label
20 day upper Bollinger band	UB(20)
20 day lower Bollinger band	LB(20)
50 day moving average	MA(50)
50 day moving average return	MA(50)_return
50 day upper Bollinger band	UB(50)
50 day lower Bollinger band	LB(50)
100 day moving average	MA(100)
100 day moving average return	MA(100)_return
100 day upper Bollinger band	UB(100)
100 day lower Bollinger band	LB(100)
200 day moving average	MA(200)
200 day moving average return	MA(200)_return
200 day upper Bollinger band	UB(200)
200 day lower Bollinger band	LB(200)

General Simulation Algorithm

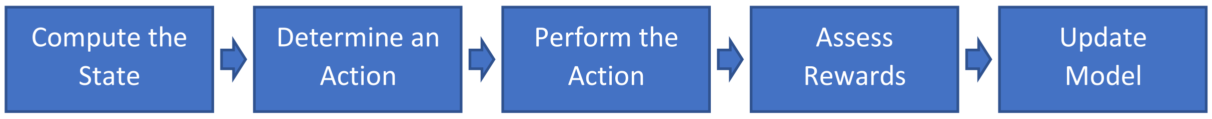
The simulation process is the same for training and testing and works as follows:

1. Initialize the portfolio to set cash = initial investment and number of shares = 0 for each stock
2. For each stock in the portfolio –
    a. For each day in the date range –
        i. Have trader make buy/sell decision based on stock indicators for the day
        ii. If buy – take all cash for the stock and buy at the current price
        iii. If sell – sell all shares for the stock and convert to cash
3. Compute the final portfolio value and net gain(loss)
4. Plot buy/sell decisions

The basic difference between testing and training is that during training a reward is calculated and the trader adjusts as appropriate to learn (see below). Note that since the algorithm goes stock by stock and then by date that initial learning is biased toward the stocks at the beginning of the portfolio. It is uncertain if this is an issue but something that should be explored later.

Learning Algorithm

Auto Trader uses a Q-learning reinforced learning algorithm (based on the Smartcab project). The algorithm executes against a state model to determine what action to take based on the inputs to the learner and the prior learning results. The action is then performed and the outcome assed to determine if the action should be penalized or rewarded. The state model is then adjusted based on the results. The basic algorithm for the learner follows this process:



Computing the State

The learning space is modeled as a dictionary of states, actions and rewards. The state in this project contains an indicator representing whether stock is owned and then a series of indicators that represent the position of the stock price on a given day relative to the features/indicators being used. For example, if a stock is trading at $80 and the 20 day moving average (MA(20)) is $74, the state indicator for MA(20) will be "above". If the trading price was $70, the indicator would be "below". "equal" is also a valid indicator (although I never saw this in practice). With all of the moving averages and bands in play this results in each possible state having 13 elements: one for the "has stock" indicator (true or false), one for each moving average (20,50,100,200) and one for each upper and lower band associated with each moving average.

While learning, Auto Trader evaluates if a state has been used before. If not it adds it to the learning model. This way only states that are actually encountered become part of the model.

Determining the Action

The possible actions for each state are "buy", "sell" and Nothing. During initial training, Auto Trader will take random actions when a state is encountered. The parameter *epsilon* controls if a random action is chosen or if a previously learned action is used. *epsilon* is coded to be 0.998^t where t is the number of data points the trader has been given to learn. During each evaluation, *epsilon* is compared to a random number between 0 and 1. If the random number is below *epsilon* a random action is taken. The choice of *epsilon* was somewhat arbitrary using observations of training results with five stocks over a one year training set. This needs to be explored further to determine if there is a formulaic way to determine how many random actions are needed to fill out the model before the algorithm relay on the learned data.

If not using a random action (*epsilon* is low or in test mode), Auto Trader picks the action for a given state with the highest reward (explained later). In the case of a tie, Auto Trader randomly picks one of the actions with equivalent reward scores.

Performing the Action

Once an action is determined, Auto Trader executes the buy/sell/hold action if possible. If the action is buy, and Auto Trader doesn't already own the stock, the following steps will be executed:

1. Buy as much of the stock as possible given the cash available for the stock: $shares = cash\ /\ stock\ price$
2. Set cash for the stock to $0
3. Record the price the stock was bought at

If the action is sell, and Auto Trader owns the stock, the following steps are taken:

1. Sell as much of the stock as possible: $cash = shares \times stock\ price$

2. Set shares to 0

Note that for the sake of simplicity, the transactions are modelled as “all or nothing” meaning that all of the shares are sold or all of the cash is used to buy. A more complex trader might buy or sell partial lots of the stock. Also, each stock has its own investment or budget without cross-over. The trader can’t use proceeds from the sale of one stock to buy another stock. Again, this is to keep the model simple but is an area of refinement that could be explored further.

Calculating Rewards

Auto Trader is rewarded (or punished) based on the outcome that would be achieved had the trader executed the chosen action in the real market. I use the daily return for MA(20), the 20 day moving average, as the reward score (more on this selection later), adjusting it based on the action taken. Using

$$MA(20)\_return = (MA(20)_{t+1} \;/\; MA(20)_{t}) - 1$$

the logic is as follows:

1. If action is buy
    a. If we already own the stock, asses a heavy penalty (-10)
    b. else set the reward to MA(20)_return
2. If action is sell
    a. If we don’t own the stock, assess a heavy penalty (-10)
    b. else set the reward to - MA(20)_return
3. If action is Nothing
    a. If we own the stock, set reward to MA(20)_return
    b. else, set reward to - MA(20)_return

The table below illustrates how this logic works with some examples:

Own Stock?	$MA(20)_{today}$	$MA(20)_{tomorrow}$	MA(20)_Return	Action	Reward	Rationale
Yes	$100	$105	+0.05	Sell	-0.05	Should have held. Price went up.
Yes	$100	$95	-0.05	Sell	+0.05	Good choice! Price went down.
Yes	$100	$105	+0.05	Nothing	+0.05	Good choice! Price went up.
Yes	$100	$105	+0.05	Buy	-10.00	What are you thinking? You already have it.
No	$100	$105	+0.05	Buy	+0.05	Good choice! Price went up.
No	$100	$95	-0.05	Buy	-0.05	Shouldn’t have bought. Price went down.
No	$100	$105	+0.05	Nothing	-0.05	Should have bought. Price went up.
No	$100	$105	+0.05	Sell	-10.00	Can’t do that! You have nothing to sell.

Learning and Updating the Model

Once rewards are calculated, the model is updated using the following learning formula:

$$Q(state, action) = (1 - \alpha) \times Q(state, action) + \alpha \times Reward$$

Where α (alpha) is the learning factor. I used a learning factor of α = 0.5, again somewhat arbitrary and an area that could be explored further.

Note that, like the Smartcab project, this formula does not consider future rewards. This also is a potential refinement and area for further investigation.

Output and Visualizations

Auto Trader produces several data and visual outputs including:

1. Comparison of the beginning and ending portfolio values
2. Net gain(loss) achieved by Auto Trader
3. Net gain(loss) that would have been achieved by buy-hold
4. Graph of the buy/sell decisions
5. A textual representation of the policies learned by the trader
6. Graphs of the learning process – epsilon, average reward, number of q states.

Benchmark

As stated previously, the objective is to build a trader that can out-perform a simple buy and hold strategy. Would you gain more or lose less if you followed Auto Trader's recommendations as opposed to buying a stock at the beginning of a period and selling it at the end? The performance of Auto Trader is evaluated against this buy-hold benchmark.

Methodology

Data Preprocessing

After historical stock prices are retrieved from Yahoo! Finance, there is minimal pre-processing done on the data other than computing the derived features such as moving averages and Bollinger Bands described above. The only important data step is filling in missing data. It is possible that some stocks may not have been traded on all days of other stocks in the portfolio. To handle this situation, the code fills in missing prices. It does this by first doing a "forward fill" where a missing value gets the value from the prior day and then a "back fill" where a missing value get set to the value of the next day. This two part action makes sure missing values are filled in not only in the middle of the range but both ends of the range as well.

Implementation

The Auto Trader was coded in Python utilizing code from previous Machine Learning Nanodegree projects (Smartcab and Customer Segmentation) as well as code from the Machine Learning for Trading course. There are three main modules for the code. *trader.py* contains the main code segments including the setup, learning algorithm and simulation. *data_utilities.py* contains the code for reading and processing the stock data. *visuals.py* contains code for plotting results of the simulation. A Jupyter Notebook (*stock trader.ipynb*) was used for data exploration, analysis and capturing results from running the code (can also be run from the command line).

An iterative approach was taken to coding, testing and evaluating Auto Trader. I started with a single stock (*Apple*) and a single feature (*MA(20)*). The code and analysis was then expanded to handle multiple stocks and features. Diagnostic output and graphs were added along the way to better understand how Auto Trader was performing.

Refinement

Rewards

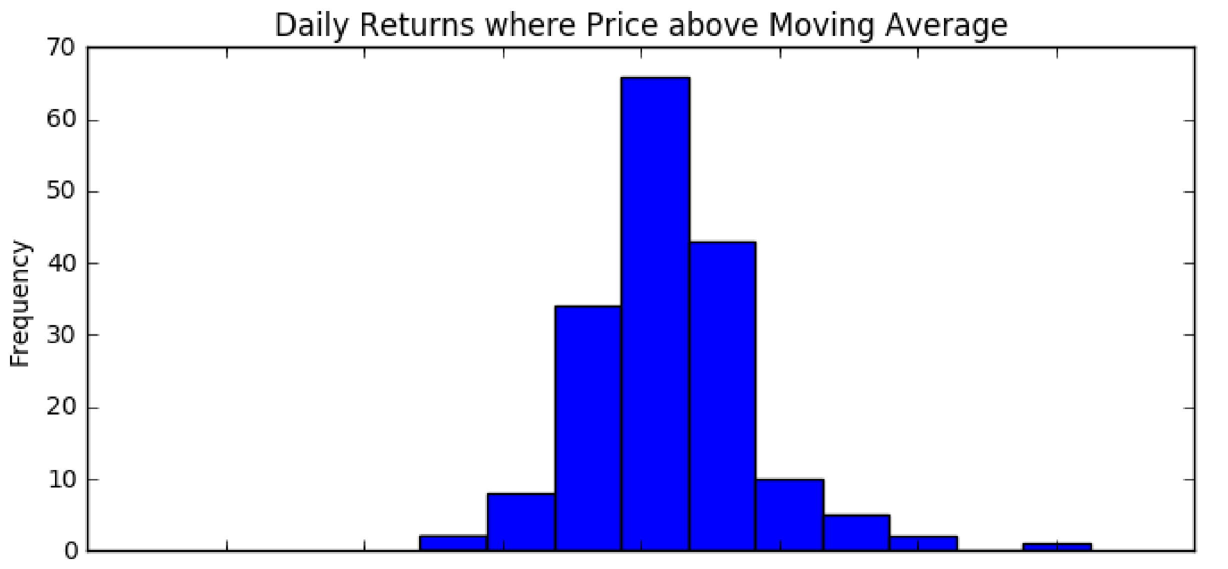
Early results in running Auto Trader were disappointing. Even in the simple one stock one feature case, the learning algorithm would produce policies that were not intuitive. Here are the resulting policies from an early run:

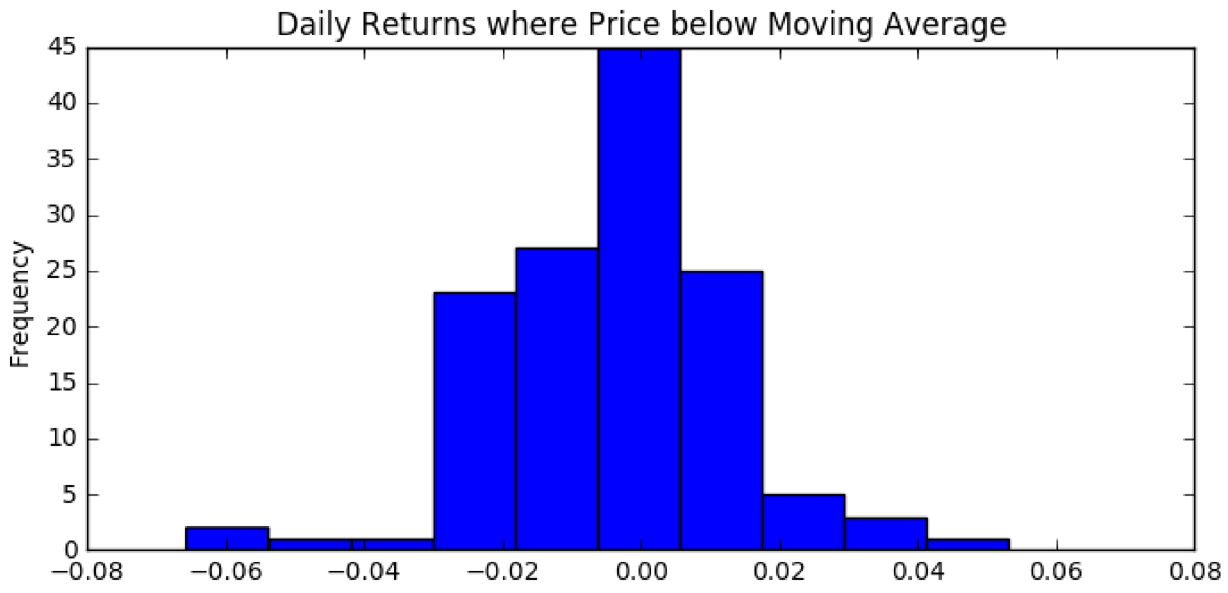
```
If has_stock is True and price is below moving average(20) then [None]
If has_stock is False and price is above moving average(20) then ['buy']
If has_stock is True and price is above moving average(20) then [None]
If has_stock is False and price is below moving average(20) then [None]
```

Note that no policy leads to a *Sell* decision and the action *None* is preferred.

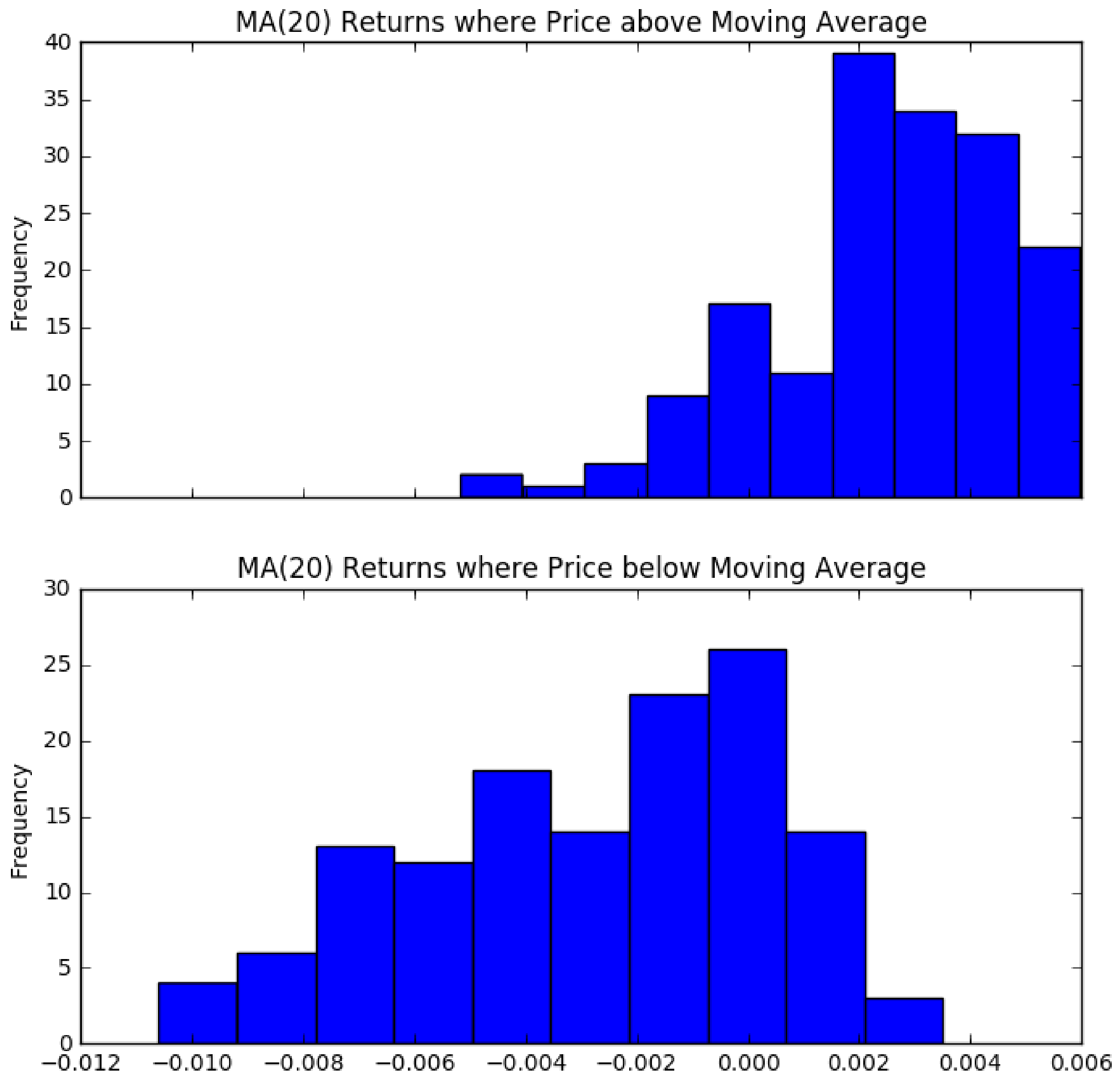
After some time of investigation and reflection on why this might be, I started to focus on the reward calculation. Originally, I was using the Daily Return of the Adjusted Close Price. My expectation was that if the right buy decision was made, the daily return would be positive and thus the reward positive and if the right sell decision was made, the daily return would be negative and thus the reward positive as well. Looking further at the data though I discovered that the Daily Return of the Adjusted Close Price was not a good discriminator of whether the decision was successful.

The graphs below show the distribution of Daily Returns where the stock price is above and below the 20 day moving average. You can see from the graphs that while the returns where the price is above the moving average skews slightly positive and the returns where the price is below the moving average skews slightly negative, the shift is relatively small. For this reason, the learning algorithm had difficulty converging on a consistent set of actions.





After finding this I looked at the Daily Returns of the 20 Day Moving Average. Those proved to be better discriminators of success and failure for the selected actions. The graphs below show the above/below distributions for the change in moving average (as opposed to the change in daily price above).



After changing the reward function to use returns based on the moving average, I got results more in line with expectations. Here are the policies based on the refinement:

```
If has_stock is True and price is below moving average(20) then ['sell']
If has_stock is False and price is above moving average(20) then ['buy']
If has_stock is True and price is above moving average(20) then [None]
If has_stock is False and price is below moving average(20) then [None]
```

Epsilon

As stated previously, *epsilon* was set for what I though was good for 5 stocks and a year-long training set. Graphs were added to help visualize the impact of the changes on the learning process. This is an area that need further investigation so that *epsilon* can be parameterized rather than manually adjusted.

Features

After the code was working on a single stock and feature, the feature set was expanded to include moving average of different time windows and the Bollinger bands for those moving averages.

Results

Model Evaluation and Validation

The table below shows the trading results for 5 stocks using the 20 day moving average as the feature. The trader was trained on data from 2014 to 2015 and tested on data from 2016. As you can see, the AutoTrader outperformed buy/hold on some stocks and not others and overall for this portfolio, buy/hold provided better results.

	Cash	Shares	Initial Price	Ending Price	Buy/Hold Value	AutoTrader Value	Difference
AAPL	$ -	109.6035	$ 102.61	$ 116.23	$ 11,326.73	$ 12,738.78	$ (1,412.05)
GE	$ -	343.1328	$ 29.54	$ 31.46	$ 10,648.53	$ 10,794.64	$ (146.11)
XOM	$ -	116.7778	$ 73.45	$ 88.69	$ 12,076.07	$ 10,357.46	$ 1,718.61
DD	$ 11,938.54	0	$ 61.32	$ 73.45	$ 11,978.20	$ 11,938.54	$ 39.66
GS	$ -	57.20159	$ 173.97	$ 237.56	$ 13,654.86	$ 13,588.76	$ 66.10
					$ 59,684.40	$ 59,418.19	$ 266.21

Expanding the number of stocks to 30 (I used the stocks in the Dow Jones Average), the features to include the longer term moving averages (50, 100 and 200), and expanding the training period to cover ten years from 2005 to 2015, yielded the results in the table below.

	Cash	Shares	Initial Price	Ending Price	Buy/Hold Value	Auto Trader Value	Difference
MMM	$ -	57.02841	$ 154.67	$ 177.26	$ 11,460.01	$ 10,108.61	$ 1,351.40
AXP	$ 9,299.27	0	$ 89.25	$ 73.31	$ 8,213.49	$ 9,299.27	$ (1,085.78)
AAPL	$ -	87.59302	$ 104.27	$ 115.75	$ 11,100.36	$ 10,138.75	$ 961.62
BA	$ -	78.36632	$ 120.62	$ 153.19	$ 12,700.20	$ 12,004.90	$ 695.31
CAT	$ 12,147.55	0	$ 83.66	$ 91.76	$ 10,968.87	$ 12,147.55	$ (1,178.68)
CVX	$ -	73.78578	$ 102.13	$ 116.70	$ 11,427.17	$ 8,611.02	$ 2,816.15
CSCO	$ -	308.2628	$ 25.62	$ 29.94	$ 11,685.16	$ 9,229.35	$ 2,455.81
KO	$ -	206.0736	$ 39.15	$ 41.24	$ 10,531.77	$ 8,497.66	$ 2,034.11
DIS	$ -	114.3873	$ 91.24	$ 104.56	$ 11,459.49	$ 11,960.34	$ (500.84)
DD	$ 12,846.48	0	$ 66.01	$ 73.09	$ 11,072.47	$ 12,846.48	$ (1,774.01)
XOM	$ -	101.2263	$ 85.10	$ 88.69	$ 10,422.29	$ 8,978.14	$ 1,444.15

	Cash	Shares	Initial Price	Ending Price	Buy/Hold Value	Auto Trader Value	Difference
GE	$ -	416.2199	$ 23.30	$ 31.46	$ 13,500.40	$ 13,093.89	$ 406.50
GS	$ -	57.56125	$ 188.45	$ 237.56	$ 12,606.02	$ 13,674.20	$ (1,068.18)
HD	$ -	76.67485	$ 98.68	$ 134.28	$ 13,608.44	$ 10,296.11	$ 3,312.33
IBM	$ -	58.25787	$ 148.54	$ 163.69	$ 11,020.37	$ 9,536.46	$ 1,483.91
INTC	$ -	311.4506	$ 33.68	$ 36.13	$ 10,728.52	$ 11,253.13	$ (524.61)
JNJ	$ -	93.06193	$ 98.07	$ 114.73	$ 11,698.59	$ 10,677.07	$ 1,021.52
JPM	$ -	138.6231	$ 58.81	$ 84.93	$ 14,440.47	$ 11,773.17	$ 2,667.30
MCD	$ -	86.08017	$ 86.82	$ 121.89	$ 14,040.07	$ 10,492.55	$ 3,547.51
MRK	$ 7,500.87	0	$ 53.23	$ 58.62	$ 11,012.45	$ 7,500.87	$ 3,511.58
MSFT	$ -	162.6726	$ 44.04	$ 62.52	$ 14,197.04	$ 10,170.45	$ 4,026.59
NKE	$ 9,062.65	0	$ 46.32	$ 50.90	$ 10,990.05	$ 9,062.65	$ 1,927.40
PFE	$ -	307.0117	$ 28.64	$ 31.85	$ 11,121.82	$ 9,779.55	$ 1,342.27
PG	$ -	108.7487	$ 83.48	$ 83.05	$ 9,948.36	$ 9,031.66	$ 916.70
TRV	$ -	71.04381	$ 100.18	$ 121.62	$ 12,140.62	$ 8,640.29	$ 3,500.33
UTX	$ -	102.3122	$ 108.78	$ 109.90	$ 10,102.96	$ 11,243.71	$ (1,140.75)
UNH	$ -	61.74679	$ 97.10	$ 160.44	$ 16,522.68	$ 9,906.78	$ 6,615.90
VZ	$ -	243.7228	$ 41.89	$ 52.54	$ 12,542.95	$ 12,806.31	$ (263.36)
V	$ -	117.7674	$ 65.17	$ 78.18	$ 11,996.06	$ 9,207.20	$ 2,788.87
WMT	$ 7,653.03	0	$ 80.09	$ 68.30	$ 8,527.70	$ 7,653.03	$ 874.67
					$ 351,786.85	**$ 309,621.14**	**$ 42,165.72**

As you can see, again the buy/hold strategy performs better for the selected portfolio and test period.

Adding in the Bollinger Band features does not improve the situation. Buy/hold continues to beat the trader.

```
Net return from testing 7,511.89
Compared to buy/hold return of 51,786.85.
```

Justification

Although Auto Trader did learn trading policies given the derived features, it was not able to outperform the market using a simple buy hold strategy. It was able to outperform buy/hold for some stocks and in both rising and falling stock price situations, but the results were not consistent enough across the portfolio to claim success.

Free-Form Visualization

The Auto Trader results for Apple (APPL) from 2015 to 2016, highlights some the challenges with my approach.

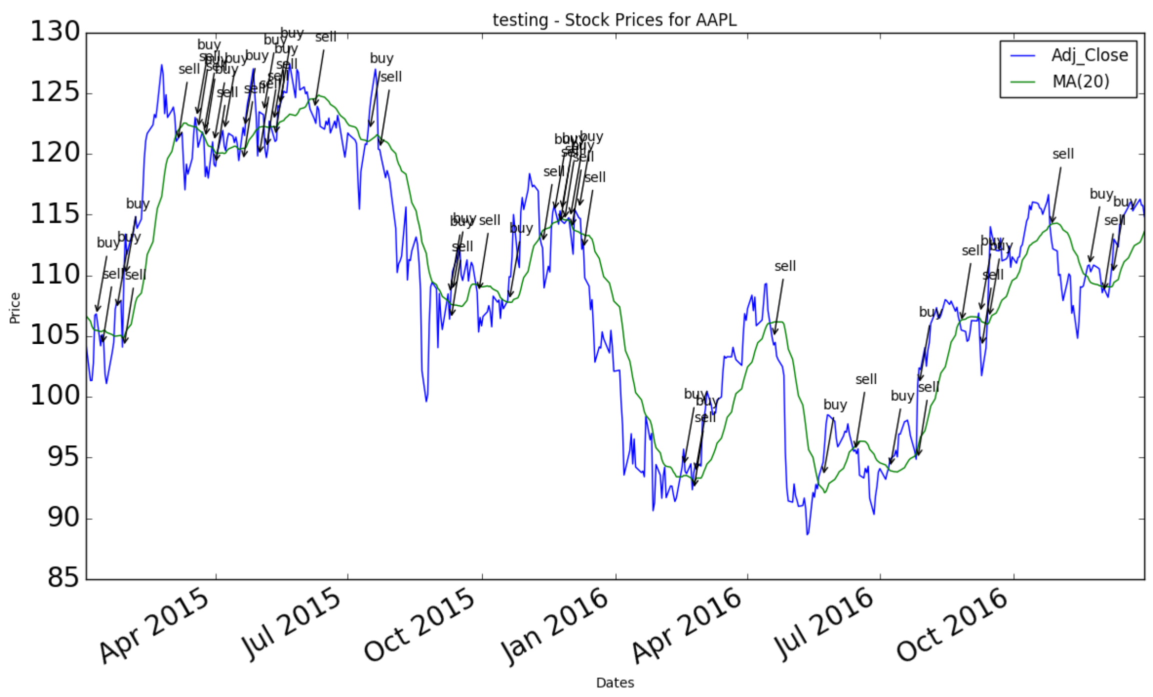


Figure 5

You can see that for 2016, the stock was somewhat "well behaved" in that once it rose above or fell below the moving average, it tended to stay that way. 2015 was not as predictable though. During 2015, the stock would rise and fall below the moving average frequently causing the trader to buy and sell more frequently with less favorable outcomes.

Reflection

For this project, I built a machine learning Auto Trader that learned buy/sell strategies using stock prices and derived stock price data. Auto Trader was able to learn and execute trading policies as expected but the effectiveness of the policies was not sufficient to consistently beat a simple buy/hold trading strategy. Along the way though I did have some key learnings.

First, I placed too much faith in the value and reliability of using moving averages to drive the trading decisions. Although the investment strategies sounded logical and my initial exploration looked promising, the data did not behave as expected often enough to make the trading polices effective. More time should have been given to evaluating the data to see if the assumptions were valid.

Next, calculating the reward for the learner may not be always be straightforward. I probably should have spent more time defining "success" for the trader. What time frame is the trader being evaluated on? 1 week? 6 months? 1 year? The fact that I benchmarked against a buy/hold strategy for a year or more yet rewarded the trader on a shorter term basis could be problematic. This needs more investigation.

Finally, using a reinforced learner like this to “uncover” trading policies might not be the most effective approach. My thought was that I could provide the learner a lot of features and it would then discover new policies, ones that I had not considered, based on the features. This is in fact how the learner works. The problem though is dependency of the polices on the data provided. I assumed that the most effective polices would come from changes in prices relative to the various moving average therefore that’s the data I provided. My basic assumption proved to be wrong but since the data provided only supported that assumption, the learner was constrained and could do no better. As stated before, more analysis was needed on the data used for prediction and it’s effectiveness.

Improvement

There are multiple areas for improvement, some of which have been discussed before.

Rewards

More thought needs to be put in the reward calculation. Is the trader being rewarded for short-term gains? Long term? How does that decision impact what reward is given? For example, if the trader is to be judge on weekly returns, the reward could be derived from stock prices a week later. If judged on yearly returns, rewards could be based on prices a year later. This could have a major impact on the policies learned. Also, adding future rewards (gamma !=0) in the learning calculation should also be considered.

Trading Rules

The trader used a simple “all or nothing” rule when buying or selling stocks. This could be improved to allow the trader to buy or sell partial lots based on some level of confidence in the trading decision. Perhaps the Bollinger Bands could be used for this purpose. For example, if the stock is above the moving average but below the upper Bollinger Band spend X% cash but if it is above the Bollinger Band spend Y%. This could be accomplished by expanding the actions from simple buy and sell to buy and sell of specific quantities (e.g. buy 25%, sell 50%).

Also, for simplification, each stock had its own budget (e.g. $10,000). A more complex scheme would have a single budget amount and then determine how much to allocate to each stock. This might be able to be accomplished by expanding the states and actions to include individual investments (e.g has_stock_1, sell_stock_3). The might need to be some other indicators provided to specify how well a stock has done if we are to make tradeoff decisions between the stocks. This improvement, combine with the partial buy/sell above would obviously dramatically increase the size of the Q-table and slow down learning but it may have substantial benefit in making the trader more profitable.

Finally, the trader only buys and sell stocks at market price. Implementing the ability to leverage short-shelling may have benefit especially in market downturns.

Input Features

As mentioned before, more analysis is needed to determine what input features to use. The project used moving average and current price relative to the moving averages. Other strategies use moving averages relative to each other. Also, exponential averages should be explored as well. A methodical and analytical approach is needed to review the different features and determine which ones are best to guide the trader.