Machine Learning Engineer Nanodegree

Capstone Project

Arya Roy
29th March, 2018

I. Definition

1. Project Overview

The emergence of structured market data and improvement in the computational power in the last decade has led to machine learning getting successfully applied in quantitative trading. Finance practitioners often derisively refer to this methodology as data mining as financial data are not only quite limited (unless we use tick data), they are also not very stationary in the statistical sense as discussed in [1]. That is, the probability distribution of returns does not stay constant forever. If we use Machine Learning algorithms on these data, it is very easy to come up with trading rules that worked extremely well in certain past periods, but fail terribly going forward. However, this problem also plagues handcrafted models, but remedial action is possible. No such luck with machine-learned rules.

The major thread of the history[1] of reinforcement learning is centered on the study of trial-and-error learning that started in the psychology of animal learning. Perhaps the first to succinctly express the essence of trial-and-error learning was Edward Thorndike. The earliest computational investigations of trial-and-error learning were perhaps by Minsky and by Farley and Clark, both in 1954. In his Ph.D. dissertation, Minsky discussed computational models of reinforcement learning and described his construction of an analog machine composed of components he called SNARCs (Stochastic Neural-Analog Reinforcement Calculators). Farley and Clark described another neural-network learning machine designed to learn by trial and error. In the 1960s the terms "reinforcement" and "reinforcement learning" were used in the

---

[1] Source : http://www.incompleteideas.net/book/ebook/node12.html

engineering literature for the first time (e.g., Waltz and Fu, 1965; Mendel, 1966; Fu, 1970; Mendel and McClaren, 1970). Particularly influential was Minsky's paper "Steps Toward Artificial Intelligence" (Minsky, 1961), which discussed several issues relevant to reinforcement learning, including what he called the credit assignment problem: How do you distribute credit for success among the many decisions that may have been involved in producing it? All of the methods we discuss in this book are, in a sense, directed toward solving this problem.

Despite the reluctance to use AI techniques in trading, this is a rapidly advancing field, where researchers are working towards developing techniques to solve the overfitting bias as mentioned in the previous paragraph. One such method is the use of a reinforcement learning framework to generate profitable trading signals. One key advantage of this method over supervised learning approaches is does not require learning from examples provided by a knowledgeable external supervisor [2]. In order to use the reinforcement learning algorithm, I feed my learning agent with market data from the National Stock Exchange[2] database for all the tickers in the Nifty 50 index for 1820 trading days. There is also a testing dataset consisting of 190 most recent trading days that is used to find out how the learning agent performs in out-of-sample cases.

The dataset used in this project is also known as Bhavcopy[3] – an archive maintained by the NSE. Bhavcopy report is the trade data report of a trading day. It contains the opening price, closing price, previous closing price, highest trade price of the day, lowest trade price of the day, 52 week yearly highest trade price, 52 week yearly lowest price, total trade volume, total number of trades and total traded value of all the securities that were traded in a day. It is generated on every trading day 10 minutes after market closes. It is used by traders to understand the behavior of all the traded securities for the day.

2. Problem Statement

The goal of the project is to develop an adaptive learning agent that trains itself based on historical price data on how to maximize the risk-adjusted return (including net transaction cost)

---

[2] Source : https://www.nseindia.com/

[3] Source : https://www.nseindia.com/products/content/equities/equities/archieve_eq.htm

i.e. Shape ratio. The environment used here is the historical closing prices of 50 large-cap stocks listed in the National Stock Exchange (Nifty 50 - ^NSEI) ranging from 4th Jan, 2010 to 21t Feb, 2018. The states of the agent are defined based on whether the returns are above, within or below a dynamic price momentum band around the moving average of return (window of 12 trading days). On each trading day, the agent must generate a valid action – buy stock, do nothing or sell stocks. Based on its actions, the agent should receive a reward or penalty equal to the risk-adjusted stock returns (profitable or loss making) and the transaction cost as a function of the closing price for every new buying or selling of a stock. Based on the rewards or penalties it gets or each stock, the agent should learn an optimal trading strategy to maximize the risk-adjusted returns of a portfolio of stocks. In order to learn from actual trading data can be costly as it involves the possibility of making losses while the agent is learning. Hence I have used a novel method called Dyna Q as given in [2] in order to solve the problem of insufficient market data. Using Dyna Q algorithm, the real experiences, passing back and forth between the environment and the policy, affect the policy and the value functions in much the same way as simulation generated by the model of the environment.

3. Metrics

The 3 metrics used to evaluate the performance of the proposed model are: Sharpe Ratio, Maximum Drawdown and Maximum Drawdown Duration.

According to [8], Sharpe Ratio is defined as the excess return over the risk-free rate per unit of underlying risk. Here the proxy for portfolio risk is volatility or standard deviation of returns. Here the daily volatility of returns is calculated as opposed to the weekly or monthly one to give more conservative Sharpe ratio.

$$S = \frac{Portfolio\ Return - (Risk-free\ Rate)}{Standard\ Deviation\ of\ Portfolio\ Return}$$

In the above equation, S is the Sharpe Ratio. The higher the Sharpe Ratio, the higher the expected risk-adjusted return from the portfolio. The next metric used is Maximum Drawdown. According to [9], Maximum Drawdown is the indicator of the downside risk associated with a particular stock or portfolio. It is defined as the maximum loss from peak to trough of a portfolio before the portfolio returns are positive again. The third metric used is the Maximum Drawdown Duration. This is simply the duration in days when the portfolio value falls for Maximum Drawdown. Hence for the Maximum Drawdown period,

$$\text{Maximum Drawdown} = \frac{(Peak\ Vale - Trough\ Value)}{Peak\ Value}$$

We will find out if the proposed model provides healthy returns (daily Sharpe ratio above 0.2) in different scenarios. We will test the robustness of the model by testing the optimal policy learned over 5 different testing datasets with 50 trading days included in each.

We will observe the daily Sharpe ratio to decide whether the proposed model gives a significant risk-adjusted return in the 5 different testing periods. In the period with the least risk-adjusted return, we will check the maximum drawdown and the maximum drawdown duration to ascertain how capital-intensive the trading strategy is.

II. Analysis

1. Data Exploration

The dataset consists of market data (open, high, low, close, volume and open interest) for all the stocks listed in the Nifty 50 for every trading day ranging from 4th January, 2010 till 21st February, 2018 (a total of 2012 trading days) as shown in Figure 1.

```
>>> import os
>>> from fileinput import filename
>>> def explore() :
...     base='E:/Bhavcopy/data/NSE-EOD'
...     filenames=os.listdir(base)
...     print ("Number of files:",len(filenames))
...
>>> explore()
Number of files: 2012
```

Figure 1: Code snippet showing number of files

The start date is chosen such that the data excludes the 2008-09 recession. This data is sourced directly from the National Stock Exchange database using an open-source application [3]. There are files corresponding to each trading day with a list of tickers. The sample of 1 file is shown in Table 1 below:

<ticker>	<date>	<open>	<high>	<low>	<close>	<volume>	<o/i>
20MICRONS	2/21/2018	51.75	52	50.05	50.35	46224	24176
21STCENMGM	2/21/2018	45.15	45.15	43.45	43.45	10217	0
3IINFOTECH	2/21/2018	5.75	5.85	5.6	5.7	2910036	1550093
3MINDIA	2/21/2018	20799	20906	20430	20543.45	1446	682
5PAISA	2/21/2018	333.15	350.6	333.15	350.6	8535	7001

Table 1: Trading data in the imported file fpr 2/21/2018

The <ticker> column corresponds to the tickers that are traded in the market on the particular day. The <date> column gives the date on which the trade is executed. <open> column gives the price at which the ticker commenced trading on the exchange on the given day. <high> column gives the highest price of the ticker on the given day. The <low> column gives the lowest price of the ticker on the given day. <close> column gives the last price at which the stock traded on the particular day. The <volume> column gives the total number of shares traded on that day. The <o/i> column gives the total number of outstanding contracts at the end of the day.

There are total 764042 missing points in the final dataset including all the trading days. These data points are missing mainly because the specific tickers were not traded on the given trading days. However our concern is the large cap stocks for which the number of missing data points is only 6130.

An interesting observation is that there are certain market participants that are very receptive to the VIX[3], a trademark of Chicago Board Options Exchange, Incorporated ("CBOE") and Standard & Poor's has granted a license to NSE, with permission from CBOE, to use such mark in the name of the India VIX and for purposes relating to the India VIX. India VIX is a volatility index based on the NIFTY Index Option prices. On the day when the VIX was its global maximum, the z-score of the market participants was calculated.

A z-score indicates how many standard deviations an element is from the mean. A z-score can be calculated from the following formula:

$$z = (X - \mu) / \sigma$$

where z is the z-score, X is the value of the element, μ is the population mean, and σ is the standard deviation. A higher z-score indicates higher deviation from the mean and vice-versa.

---

[3] Source : https://www.nseindia.com/live_market/dynaContent/live_watch/vix_home_page.htm

Top 3 absolute z-score		Bottom 3 absolute z-score	
Ticker	Absolute z-score	Ticker	Absolute z-score
NIFTY50DIVPOINT	1.838677	NIFTYGROWSECT15	0.011092
NIFTYREALTY	0.780695	NIFTYPHARMA	0.032290
NIFTYPSE	0.740699	NIFTYQUALITY30	0.060581

Table 2: Z-score of index tickers on maximum volatility trading day

The tickers used in Table 2 are explained in Table 3.

Ticker	Explanation

NIFTY50DIVPOINT	Designed to track the total dividend from the constituents of the NIFTY 50 index
NIFTYREALTY	Designed to reflect the behaviour and performance of Real Estate companies
NIFTYPSE	Comprises of 20 stocks that are listed on the National Stock Exchange (NSE) where 51% of company's outstanding share capital is held by the Central Government and/or State Government, directly or indirectly.
NIFTYGROWSECT15	Designed to provide investors exposure to the liquid stocks from sectors of market interest.
NIFTYPHARMA	Captures the performance of the pharmaceutical sector
NIFTYQUALITY30	Index constituents are selected based on indicators like Return on Equity, Debt-to-equity ratio and average year-on-year growth in PAT in previous three years

Table 3: Information regarding tickers in Table 2

Therefore based on the above evidence, we can conclude that dividend payout, real estate stocks and public sector stocks are highly receptive to changes in VIX. On the other hand, the most liquid stocks in the major sectors, especially Pharmaceuticals and Quality stocks with superior risk-adjusted historical returns are quite resilient to changes in VIX.

2. Exploratory Visualization

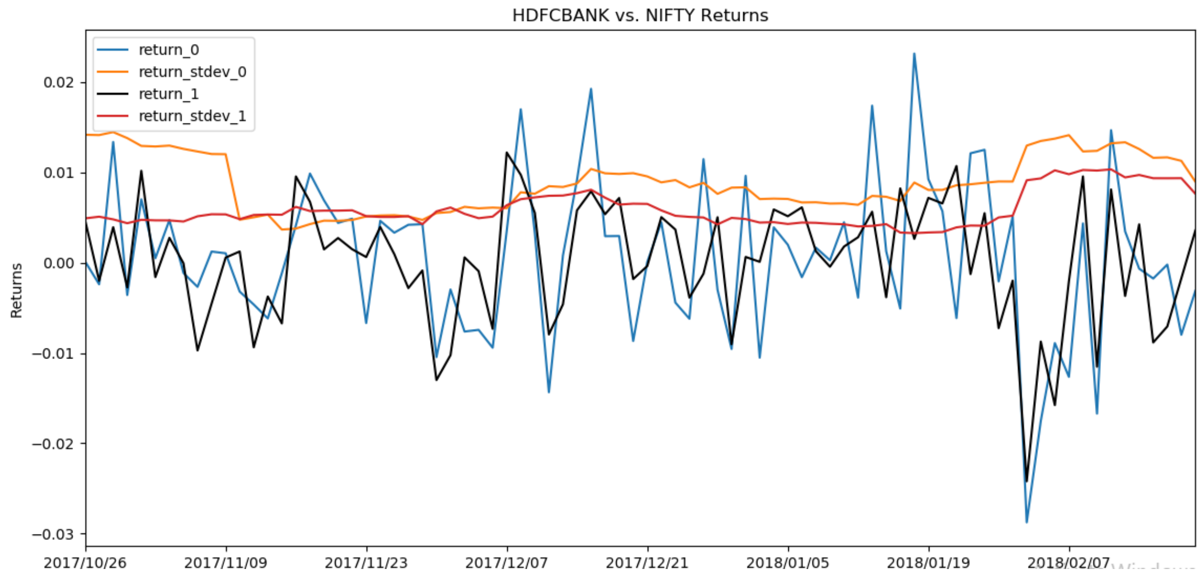

Figure 2: Plot of returns and volatility of 'HDFCBANK' and 'NIFTY'

The above figure shows the relation between the returns on a stock 'HDFCBANK' (*return_0*) and that of the exchange 'NIFTY' [1](*return_1*). The standard deviation of the 'HDFCBANK' returns is in general higher than that of 'NIFTY'. Also, it is observable from the above data that there is a strong correlation between the returns of the stock compared to the exchange.

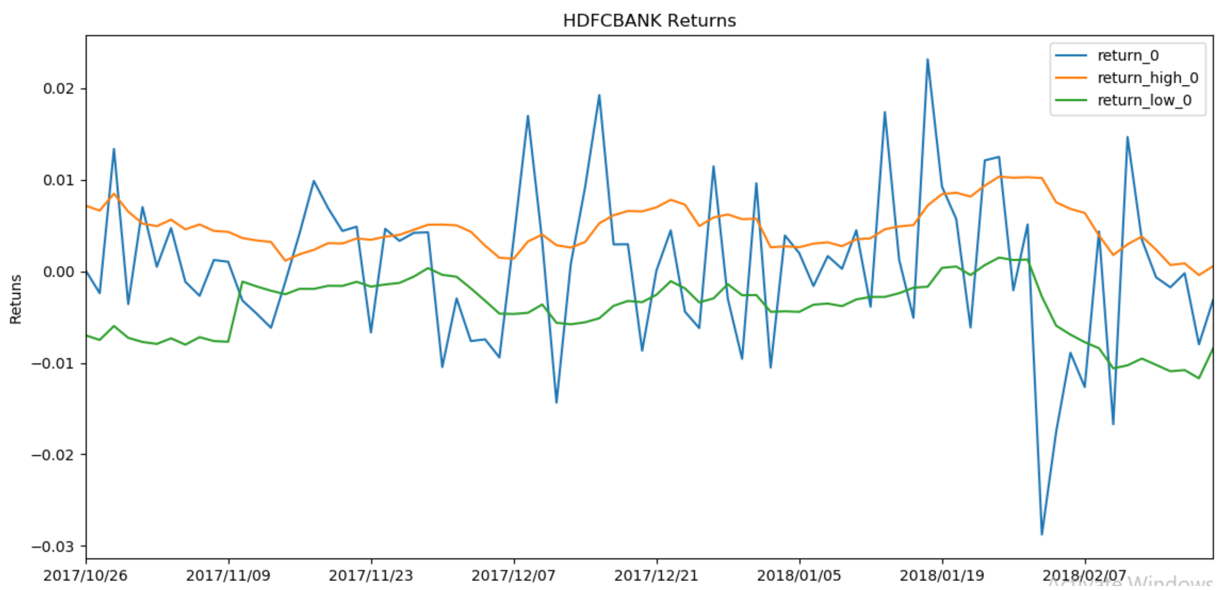

Figure 3: 'HDFCBANK' returns and the corresponding band 1 standard deviation on either side of moving average return

The above figure shows the returns of the 'HDFBANK' stock as well the bandwidth of 1 standard deviation on either side of the moving average of the returns on the stock. For the 'HDFCBANK' stock, it is in state 1 when the returns are above the orange line (*return_high* : 1 standard deviation above the moving average), state 0 when the returns are between the orange line and the green line(*return_low* : 1 standard deviation below the moving average of stock returns) and in state -1 when returns are below the green line.

Furthermore, I use the 10 year sovereign bond rates as the risk-free rate. The annual rates are converted to daily rates in order to calculate the risk-adjusted daily rates. The returns on the sovereign bonds are considered the benchmark for our model. In this project it will also be demonstrated that this agent works better than a model taking random action on the same set of stocks,

---

[1] Source : https://en.wikipedia.org/wiki/NIFTY_50

3. Algorithms and Techniques

The goal in using reinforcement learning to adjust the parameters of a system is to maximize the expected payoff or reward that is generated due to the actions of the system according to [2]. This is accomplished through trial and error exploration of the environment. The system receives a reinforcement signal from its environment (a *reward)* that provides information on whether its actions are good or bad. The performance function at time *T* can be expressed as a function of the cumulative return (product of daily returns).

$$U_T = U(R_1, R_2, \ldots, R_T)$$

Here the Markov Decision Process (MDP) is used to implement Reinforcement Learning according to [4]. The MDP model consists of a discrete set of states S and a discrete set of actions A. The set of actions are as follows:

$$a_t \in (\mathit{None, Buy, Sell})$$

None indicates that the agent takes no action in the current time step. *Buy* indicates that the agent makes a *Buy* order of 1 share. *Sell* indicates that the agent makes a *Sell* order of 1 share.

At each discrete time step *t*, the agent computes its current state and takes an action a_t. Based on this, the environment provides the reward $r_t = r(s_t, a_t)$ that is the payoff due to the action taken. The environment also provides the next state $s_{t+1} = f(s_t, a_t)$.

The reward and next state functions depend only on the current state and action. Thus it does not have a memory.

The task of the agent is to learn a policy Ω that maps each state to an action $(\Omega: S \rightarrow A)$, selecting its next action a_t based solely on the current observed state s_t that is $\Omega(s_t) = a_t$. The optimal policy, or control strategy, is the one that produces the greatest possible cumulative reward over time. So, we can state that:

$$V^{\Omega}(s_t) = \mu_0 r_t + \mu_1 r_{t+1} + \mu_2 r_{t+2} + \ldots\ldots = \sum_{i=0}^{\infty} \mu_i r_{t+i}$$

$V^{\Omega}(s_t)$ is the discounted cumulative reward with the discount factor μ_i E (0,1). V represents the cumulative value achieved by following the policy Ω from an initial state s_t. μ_i represents the relative value of delayed versus immediate reward.

If we set $\mu_i = 0$, then only immediate reward is considered. As $\mu_i \rightarrow 1$, the future rewards are given greater emphasis compared to the immediate reward. In our case the value of μ_i is given below:

$$\mu_i = \sin(\exp(-i))$$

The optimal policy Ω^* that will maximize $V^{\Omega}(s)$ for all states s can be written as:

$$\Omega^* = \text{argmax}_{\Omega} V^{\Omega}(s) \quad , \forall s$$

There is no direct method to learn $\Omega^*: S \rightarrow A$ since there is no available training data of the form (s,a). The optimal policy needs to be learnt from the available immediate reward data $r(s_i,a_i)$ where I = 1,2,3…

We need to maximize $V^*(s_t)$ such that for all states s, the agent should prefer the next state s_1 over s_2 wherever $V^*(s_1) > V^*(s_2)$. We also need to learn $V^*(s)$ independently since the agent can only choose the action and not the state and it cannot perfectly predict the cumulative reward and immediate next state for every possible state-action transition.

To learn $V^*(s)$, we define a function $Q(s,a)$ such that its value is the maximum discounted cumulative reward that can be achieved starting from state s and applying action a as the first action. So we can write:

$$Q(s,a) = r(s,a) + \mu V^*(f(s,a))$$

$F(s,a)$ is the state resulting from applying action a to state s that is chosen by the optimal policy. Hence what we are trying to achieve is:

$$\Omega^*(s) = \text{arg max}_a \, Q(s,a)$$

The above equation implies that the optimal policy can be obtained with the current state a and current action a only if a maximizes $Q(s,a)$. Thus even if the agent has no knowledge of the functions r and f, it can find the optimal policy by choosing the maximum Q value.

In order to incorporate transaction cost, a fixed percentage of 1.5% is deducted from daily returns. This is because it is found that the sum of trading costs, including tax, brokerage etc. , is approximately 1.5% of daily returns.

The Q Learning algorithm requires many interactions with the real world. But this can be expensive especially in Algorithmic Trading as noted in [5]. Thus the Dyna Q method is used to hallucinate the real world experiences by randomly choosing a previously observed state s and action a taken in state s to update the Q table. This process is iterated 100 times in our model for every real time step. This ensures that the Q table converges to that corresponding to the optimum policy with limited real world data.

Initialize Q(s, a), for all s, a

Do forever:

(a) s $\leftarrow$ current (nonterminal) state (direct update in every timestep)

(b) a $\leftarrow$ ε-greedy(s, Q)

(c) Execute action a; observe resultant state, s’, and reward, r

(d) $Q(s, a) \leftarrow Q(s, a) + \mu(r + \arg\max_{s'} Q(s', a') - Q(s, a))$

(e) State Action Pair (s, a) $\leftarrow$ (s’, r)

(f) Repeat N time, (Dyna Q update):

s $\leftarrow$ random previously observed state

a $\leftarrow$ random action previously taken in s

State Action Pair (s’, r) $\leftarrow$ (s, a)

$Q(s, a) \leftarrow Q(s, a) + \mu(r + \arg\max_{s'} Q(s', a') - Q(s, a))$

Figure 4: Dyna Q algorithm

must converge to an optimal policy. Since we are using a deterministic MDP, this will be possible when the agent chooses actions in such a way that it visits every possible state-action pair infinitely often. We will also be using a stochastic component ω with a probability of 1% in order to randomly choose an action a from the available subset of actions A.

4. Benchmark

The benchmark will be a random agent that randomly chooses actions from the set A at every time step t. The goal of the learner agent is to outperform the random agent in terms of the risk-adjusted returns (Sharpe Ratio).

The available time series data is divided into 2 sections – training dataset and testing dataset. The training index consists of 1723 time series points from 2010/1/4 to 2016/12/22. The testing index consists of 283 time series points from 2016/12/23 to 2018/02/21. First I will analyze if the learning agent was able to improve its performance on the training dataset after different trials. After that, I will use the policy learned to simulate the learning agent behavior in different behavior in the testing dataset. The performance will be compared in the testing dataset for both the learning agent and the random agent.

We will also compare the performance of the learning agent against that of buying and holding 5-year sovereign bonds during the testing period. The 5-year sovereign bond is considered to have the risk-free rate of interest. The risk-adjusted return of the learning agent must be greater than the risk –free rate of return in order for this Q Learning strategy to be feasible.

III. Methodology

1. Data Preprocessing

For training and testing our learning algorithm, the data is preprocessed to create a single table with the closing price data with the ticker names (all tickers as of 16th January, 2018) as the columns and the trading dates as the rows. During this time the index has shown a net upward trend, starting from 5232.2 and ending at 1039.75. Since there is a significant change in the index in the period under consideration, the daily returns will be calculated and used for the purpose of learning instead of the closing price data to avoid bias.

Additionally, the following additional columns are calculated based on the existing data in order to compute the agent's state:

risk_adjusted : (Daily return – Bond rate)

risk_adjusted_moving : Moving average of risk_adjusted with a moving window of 12 trading days

risk_adjusted_stdev : Standard deviation of the risk_adjusted data in the moving window of 12 trading days

risk_adjusted_high : It is calculated for each stock in the portfolio as given below :

*risk_adjusted_moving + 1.5*risk_adjusted_stdev*

risk_adjusted_low : It is calculated for each stock in the portfolio as given below :

*risk_adjusted_moving – 1.5*risk_adjusted_stdev*

Now based on the above calculated data, the state of the individual stock is decided:

-1 : stock return is below *risk_adjusted_low*

0 : stock return is between *risk_adjusted_low* and *risk_adjusted_high*

1 : stock return is above *risk_adjusted_high*

```
>>> df1 = pd.read_csv('E:/Bhavcopy/data/final.csv')
>>> df1['Date']= pd.to_datetime(df1['Date'])
>>> df1['Date'] = df1['Date'].dt.strftime('%Y/%m/%d')
>>> df1 = df1.set_index('Date')
>>> df1 = df1.filter(items=['HDFCBANK','NSENIFTY'])
>>> l=len(df1.columns)
>>> for i in range(l):
...     df1['_'.join(['return',str(i)])]=df1.iloc[:,i].rolling(window=2).apply(lambda x: x[1]/x[0]-1)
...     df1['_'.join(['return_moving',str(i)])]=df1.iloc[:,-1].rolling(window=12).apply(lambda x: x.mean())
...     df1['_'.join(['return_stdev',str(i)])]=df1.iloc[:,-2].rolling(window=12).apply(lambda x: x.std())
...     df1['_'.join(['return_high',str(i)])]=df1.iloc[:,-2]+0.5*df1.iloc[:,-1]
...     df1['_'.join(['return_low',str(i)])]=df1.iloc[:,-3]-0.5*df1.iloc[:,-2]
...
>>> df1.iloc[:,[2,4,7,9]].loc['2017/10/26':'2018/02/21'].plot()
```

Figure 5: Code snippet showing calculation of additional fields

2. Implementation

As given in [2], learning the optimal policy corresponds to learning the optimal Q function. The optimal state-action value function Q^* is defined as the expected return for taking the action a ∈ A at the state s ∈ S, for all (s,a) ∈ SxA. So it can be written as:

$$V^*(s) = \text{argnax}_{a'}\ Q(s,a')$$

Using the above relation, we can derive the recursive definition of the Q function such that:

$$Q(s,a) = r(s,a) + \mu\ \max_{a'} Q(f(s,a),a')$$

Thus the agent does not actually know the Q function. It can only estimate the value of the Q function, $\hat{Q}$. The agent will construct the Q-table that contains the value of $\hat{Q}$(s,a) for every pair (s,a) - the current approximation of Q(s,a). In our case, the Q table is initiated with zeros. But it can also be initialized with random numbers. At every time step t, the agent should observe the current state and update the Q-table.

One drawback of this strategy is that the agent could over-commit to actions that provide a positive Q value early in the simulation. Failing to explore other actions that could present even higher values. In order to tackle that problem, a random action is chosen in 1 out of 10 simulations thus giving every action a non-zero probability of occurrence.

The ideal approach to optimize the policy is to iterate over the same dataset multiple times until it is unable to improve performance (Sharpe ratio) significantly. The policy is then tested on the same dataset to check consistency. Lastly, the policy learned will be tested in the testing dataset.

Once the training session is over, the open positions of the learner will be closed so that the agent always starts a new position without carrying previously held positions.

The key challenge faced was to upgrade the base model used to trade a single stock to the current learning agent that trades a portfolio of stocks simultaneously. An easier implementation was to run the single stock model over the list of stocks in the portfolio but that would increase the processing time exponentially.

The next challenge was to find the optimum state representation which would be sufficient to learn an optimum trading policy within finite iterations. If the number of states is greater than 3 then learning the optimal policy will take many more iterations. Here the momentum indicator is used as a metric to decide the market condition by the learning agent. However, there many more sophisticated ways that the learning agent can evaluate the market. However, these state representations are not feasible for the learning agent as it fails to converge to an optimum policy since all the possible states in a complicated model will not be revisited sufficient number of times within 100 iterations to get an accurate estimate of the reward function. Establishing the environment was also difficult. At every time step, the agent has to evaluate which state it is in based on the environment parameters. During the training period, the agent must calculate the immediate reward for taking an action to move to the next state. Based on this, the agent is able to estimate iteratively the Q value or expected value of taking a trading action to buy, sell or hold.

The final challenge was the Dyna Q implementation to ensure convergence to the optimal policy. Although it increased the processing time by a few minutes, this was only a small price to pay for the right approximation of the Q table corresponding to the optimal policy.

3. Refinement

The aim is to tune the model such that it maximizes the Sharpe ratio [8]. The first parameter that is tuned is the bandwidth around the moving average. The charts of the daily returns of the model with varying bandwidths are shown in the following figure:

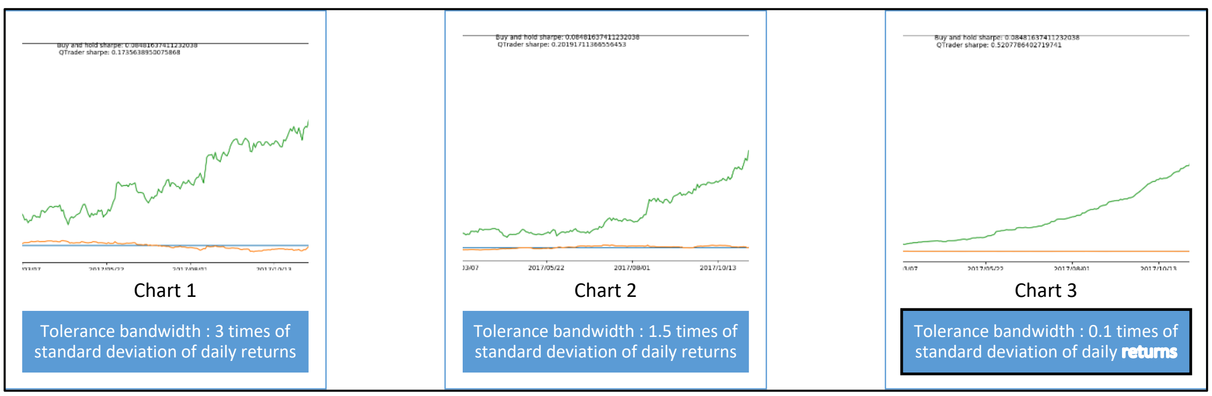

Figure 6: The daily portfolio return in the test period for different vlatility bandwidths around the moving average of daily returns

The following table shows the output parameters corresponding to different bandwidths:

	Chart 1	Chart 2	Chart 3
Annual Sharpe Ratio	2.76	3.21	8.27
Maximum Drawdown (in %)	16.1	12.2	9.5
Maximum Drawdown Duration (in days)	95	177	44

Table 4: Ouput Parameters of the model with varying tolerance bandwith around the moving average of daily returns

Table 2 shows that the best risk-adjusted returns are achieved when the bandwidth is equal to 0.1 times the standard deviation of daily returns on either side of the moving average of daily returns. Keeping this parameter fixed, the parameters directly used in the Q Learning algorithm are tuned now – alpha and discount factor.

Here alpha is the learning rate i.e. the higher the value of alpha, the lesser weight will be given to the existing value of the reward function of the learning agent prior to iteration. Therefore the higher the value of alpha, the more sensitive the reward function to new iterations. This is an important factor for the trading agent as higher learning rate will ensure that it is more responsive to the more recent market characteristics – a desirable characteristic. The default value of alpha is 0 i.e the existing value of the reward function is given the same weightage as that in the new iteration. As expected, we find that higher the value of alpha, the better the performance of the learning agent as shown in Table 3 below:

	alpha = 0.1	alpha = 0.5
Annual Sharpe Ratio	20.98	22.17
Maximum Drawdown (in %)	1.1	1.3
Maximum Drawdown (in days)	6	5

Table 5: Output parameters of the learning agent varying alpha

Lower values alpha might give better results however the learning agent will not converge if the market volatility is high. Hence alpha value is taken as 0.5. Now the final parameter that we will tune is the discount factor. This is the key factor that ensures the convergence of the Q table values. The default discount factor is taken as a constant equal to 0.9. Now we take the discount factor as sin(exp(-i)), where *i* is the number of days from the present the iteration is taking place in. The dynamic discount factor chosen here gives more weight to more recent iterations compared to earlier ones. As a result the learning agent is updated with the latest market information rather than following characteristics that are way behind in the past to contribute to trading decision currently. As expected, choosing a dynamic discount factor gives the best Sharpe ratio so far as shown in the following table:

	discount factor = 0.9	discount factor = sin(exp(-i))
Annual Sharpe Ratio	22.17	25.82
Maximum Drawdown (in %)	1.3	1.3
Maximum Drawdown (in days)	5	3

Table 6: Output parameters of learning agent varying the discount factor

Thus parameter tuning has led to 9.5 times improvement in the Sharpe Ratio from 2.76 to 25.82. While calibrating the model, special care was taken to ensure the model was not overfitted to the testing period.

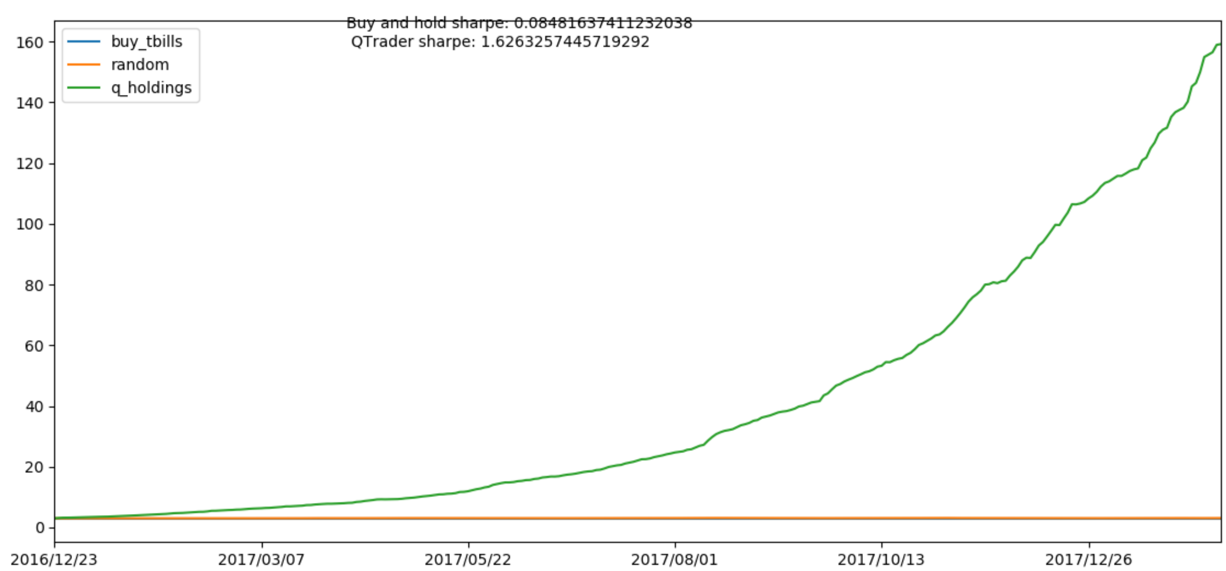

Figure 7: Daily returns of model with bandwidth = 1.5, alpha = 0.5, discount factor = sin(exp(-i))

In Figure 9, the daily returns of the final model is shown during the testing period.

IV. Results

In this section, I will evaluate the final model, test its robustness and compare its performance to the benchmark established earlier.

1. Model Evaluation and Validation

We will find out if the proposed model provides healthy returns (daily Sharpe ratio above 0.2) in different scenarios. We will test the robustness of the model by testing the optimal policy learned over 5 different testing datasets with 50 trading days included in each. The following figure shows that how the model performed in different scenarios.

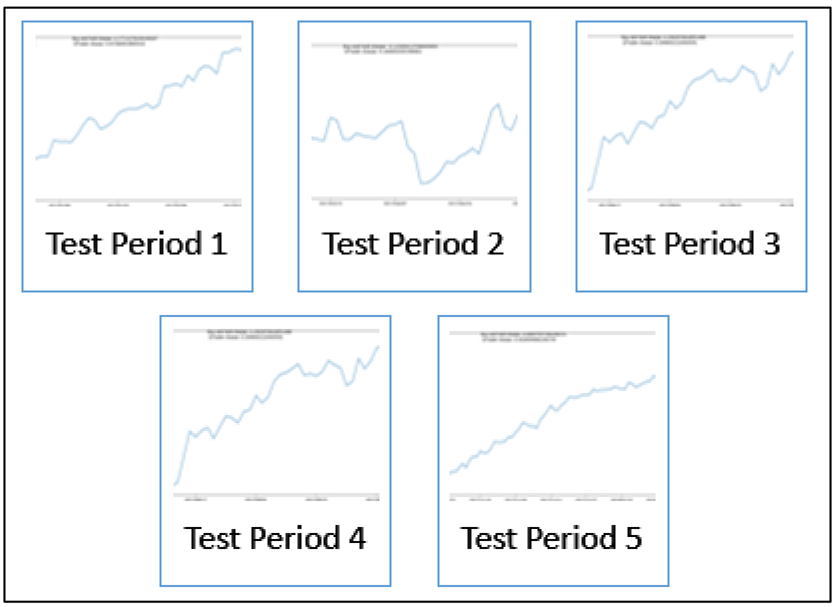

Figure 8: Returns in 5 testing periods

The model gives a significant risk-adjusted return in the 5 different testing periods after getting trained using the same dataset – daily Sharpe ratio in the range of 0.18 to 0,5). In the period with the least risk-adjusted return, the maximum drawdown is only 5 % and the maximum drawdown duration is 20 trading days.

The performance analysis for all 5 test cases is given below:

Test Period	Maximum Drawdown (in percentage)	Maximum Drawdown Duration (in days)	Annual Sharpe Ratio
2016/12/23-2017/03/06	2.8	17	7.27
2017/03/08 – 2017/05/19	5.1	20	2.93
2017/05/23-2017/07/31	3.9	19	4.91
2017/08/02-2017/10/12	1	4	7.06
2017/10/16-2018/02/14	3.1	25	8

Table 7: Test period Performance Metrics

2. Justification

The performance of the final model will be now compared to that of a random agent in the testing period from 2016/12/23 to 2018/02/14.

The optimal policy is fixed when it is run on the testing dataset. Hence the performance of the learning agent remains fixed over a testing database irrespective of the number of trial runs. However, the random agent has different portfolio holdings over the same testing database in different trial runs. Hence the average performance of the random agent in 20 trial runs is used for comparison purpose so that the comparison does not produce highly optimistic or pessimistic results.

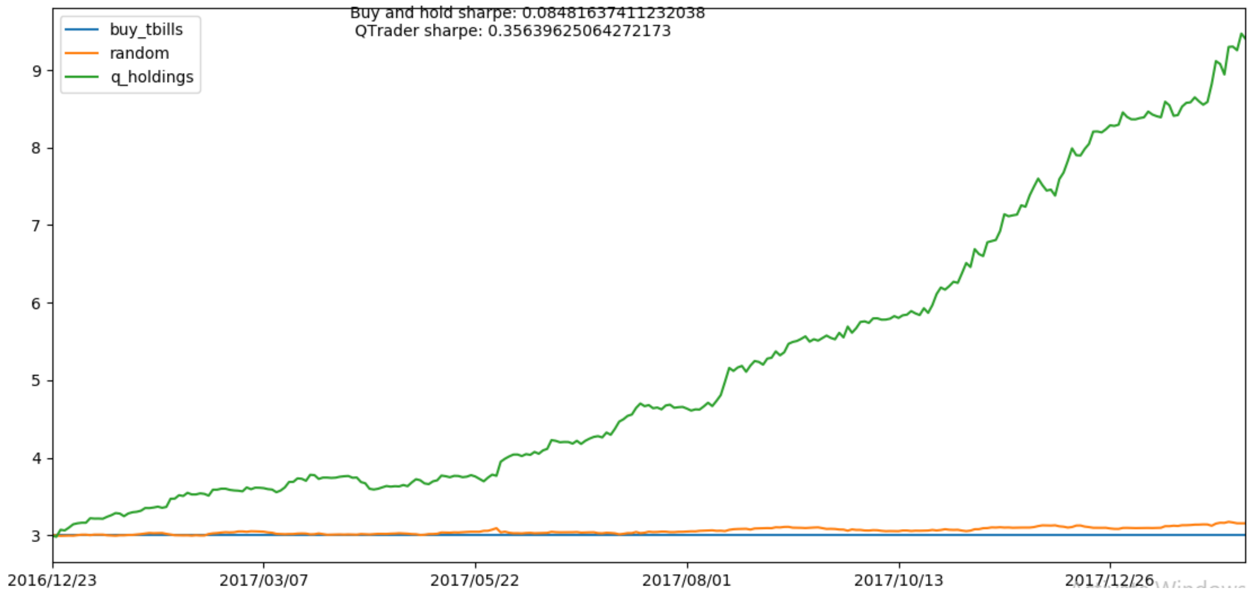

Figure 9: Comparison of daily return of learnng agent and the ranom agent against risk-free return

As can be seen in the figure above, the overall daily Sharpe ratio is significantly high (0.36). A t test is run on the time series of daily returns for the proposed model compared to the random agent. The code is shown below:

```
>>> import pandas as pd
>>> from scipy import stats
>>> df1 = pd.read_excel('E:/Bhavcopy/data/2018-02-14.xlsx',sheetname='return_tot')
>>> df2 = pd.read_excel('E:/Bhavcopy/data/2018-02-14.xlsx',sheetname='random')
>>> stats.ttest_ind(df1['q_holdings'].values,df2['random'].values)
Out[21]: Ttest_indResult(statistic=19.760623249702814, pvalue=1.9286977967445868e-66)
```

Figure 10 Code snippet showing calculation of t-statstic to show the difference in means of the learning agent cmpared to the random agent

The t-statistic for the testing period is 19.76 with a negligible p value. Therefore it is evident that the performance of the learning agent is significantly different than that of the random agent.

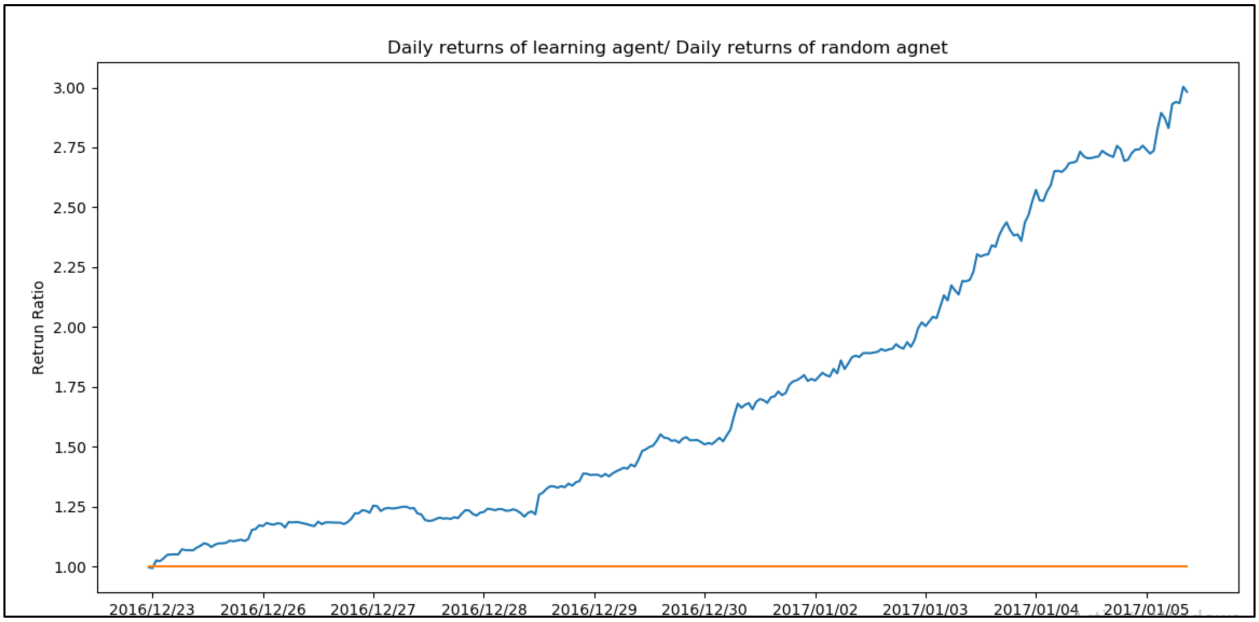

Figure 11: Ratio of daily returns of learning agent compared to the random agent

In the above figure it is evident that the ratio of the daily returns of the learning agent to that of the random agent is consistently greater than 1. The t-statistic for ratio is 20 with a negligible p value. This shows that the ratio is significantly greater than 1.This proves that the performance of the learning agent is significantly better than that of the random agent throughout the testing period.

V. Conclusion

In this section I will discuss the final outcome of the model, summarize the problem statement and solution and suggest improvements that could be made.

1. Final Remarks

In this project we have used the Reinforcement Learning Algorithm (Q Learning Algorithm) to build a trading agent that learns how to trade based on market conditions and its own states. This is a model-free approach which learns the optimal trading policy based solely on the market parameters during the testing period. The agent learns the optimal policy to maximize profits based on the market statistics part of the training dataset. It populates the Q table that lists the expected returns for a particular stock in the portfolio based on the state the agent is in. In the testing period, the learning agent references the learned Q table in order to find the next state it should be in to maximize returns. The learning agent shows a significant improvement in performance from the random agent that takes random trading decisions. The random agent is seen to give better returns than the risk-free rate of return of 5 year sovereign bonds. But this is seen when the average return of the random agent is taken after 20 trials. The learning agent gives significantly better returns than the risk-free rate since it outperforms the random agent as shown in the previous section.

One issue faced in training the learning agent was that there is a considerable chance that the agent will not converge to an optimal policy. In order to counter this problem, the agent is configured to pick a random action in 1 out of 10 training instances. This ensures that the policy is not biased towards 1 state. Also the decay parameter used falls as we go further back in time. This ensures that the expected returns approximation gives more weight to recent observations compared to older observations.

The reinforcement learning agent goes through 100 iterations in order to converge to an optimum policy. The policy learnt is used by the agent to take trading decisions during the testing period. One of the key challenges was to wrangle data to create a training dataset with the parameters required in order to represent distinct states for the learning agent. The next challenge was to represent the states in a manner such that the optimal policy can be learnt in 100 iterations or less. If the number of states is greater than 3 then learning the optimal policy will take many more iterations. As the number of states becomes significantly higher, the chances of the Q learning algorithm to converge to an optimal policy within a finite number of trials reduces. Establishing the environment was also difficult. At every time step, the agent has to evaluate which state it is in based on the environment parameters. During the training period, the agent must calculate the immediate reward for taking an action to move to the next state. Based on this, the agent is able to estimate iteratively the Q value or expected value of taking a trading action to buy, sell or hold.

The next challenge was implementing the reinforcement learning algorithm. The dataset consists of data corresponding to individual tickers. However the agent chooses a portfolio of stocks to optimize the portfolio returns. This is more complicated than using the same algorithm on a single stock.

Dyna Q implementation was necessary since the number of data points was insufficient for the convergence to the optimal policy. Although it increased the processing time by a few minutes, this was only a small price to pay for the right approximation of the Q table corresponding to the optimal policy.

2. Improvement

In this problem there are only 3 states. If we want to be more granular in defining market states then the Q –learning algorithm is not suitable as it suffers from the curse of dimensionality. In this problem, Recurrent Reinforcement Learning algorithm is found to give much better returns than Q – learning as discussed in [6]. Moreover, RRL systems is easier to use with complex market state representations.

The simple state representation here is solely for practical purposes. We need to test with different state representations to find which one leads to better learning for the trading agent. For example, if we add the price level of the index to the state representation, then we will have 9 states instead of the current 3. Therefore it is clear that adding relevant market information to the state representation leads to manifold increase in the number of states.

The Reinforcement Learning framework will benefit if we make the trading environment as realistic as possible. Here there is a single agent interacting with the market, In reality, there are multiple agents with the same goal operating in the market simultaneously. The multi-agent factor becomes significant as the size of the trading order becomes significant or if the agent is operating in an illiquid market. In that case, the reinforcement learning should involve multiple agents as discussed in [7]. Another key improvement is to include more information to facilitate trade decisions. This could include sentiment analysis in social media to gauge public opinion of a particular stock.

9. References

[1] Chan, Ernest P. Machine Trading: Deploying Computer Algorithms to Conquer the Markets. John Wiley &Sons, Inc., 2017.

[2] Sutton, Richard S., and Andrew G. Barto. Reinforcement Learning: an Introduction. The MIT Press, 2012.

[3] Kapadia, H. (2016). GETBHAVCOPY. Mumbai: Hemen Kapadia.

[4] Xin Du, Jinjian Zhai, and Koupin Lv. Algorithm trading using q-learning and recurrent reinforcement learning

[5] Classroom.udacity.com. (2018). Udacity. [online] Available at: https://classroom.udacity.com/courses/ud501

[6] John Moody and Matthew Saffell. 1999. Reinforcement learning for trading. In Proceedings of the 1998 conference on Advances in neural information processing systems II, David A. Cohn (Ed.). MIT Press, Cambridge, MA, USA, 917-923.

[7] Lee J.W., O J. (2002) A Multi-agent Q-learning Framework for Optimizing Stock Trading Systems. In: Hameurlain A., Cicchetti R., Traunmüller R. (eds) Database and Expert Systems Applications. DEXA 2002. Lecture Notes in Computer Science, vol 2453. Springer, Berlin, Heidelberg

[8] Adam Hayes, C. (2018). Sharpe Ratio. [online] Investopedia. Available at: https://www.investopedia.com/terms/s/sharperatio.asp

[9] Elvis Picardo, C. (2018). Maximum Drawdown (MDD). [online] Investopedia. Available at: https://www.investopedia.com/terms/m/maximum-drawdown-mdd.asp