A Q-learning Agent for Automated Trading in Equity Stock Markets

Link for the codes : JEEVAN-REJI/RL-PROJECT: A Q-learning Agent for Automated Trading in Equity Stock Markets (github.com)

INTRODUCTION:

The main motivation for this project is to create an automated trading agent which will use Algorithmic trading to manipulate data obtained from a particular stock and use a self improving dynamic strategy(through Reinforcement Learning Concepts)based on current stock market patterns to maximize profits obtained. At each trading instance we will use the most optimum strategy available and this strategy will also be updated to incorporate changes in the stock market patterns. Before we proceed,as in all Reinforcement Learning problems, we must clearly define our States, Actions and Rewards. The way we computed rewards is expressed in the report. Defining the states however and the subsequent Environment which it encompasses is more challenging and in this project we have deployed two methodologies to define the states. In the first method,we defined a more exhaustive and comprehensive state environment - Clusters to represent a state. We hypothesize that the trading session pattern of the stock market can be similar to a trading session of the past and then, using this assumption, we seek to find what was the most optimal trading action taken when we were at the same state in the past - and we choose this action itself for the present state as well. To do so, we divide the historical trading session into n groups or clusters with all the trading sessions in a particular cluster having some similarities and trading sessions among different clusters having some dissimilarities. The realization of the clustering is done with the help of the unsupervised learning method of k- means clustering. Each of these clusters represents a STATE and the whole set of clusters the environment. Pre-processed historical trading session data in the form of the [O,H,L,C,V] tuple(discussed later) is what is fed to the k-means algorithm to create the clusters. The second method discussed uses candlestick information of a stock to represent the states of the environment. Using the candlestick information, we define a grand total of 6 STATES-UP,HUP, VHUP,DOWN ,HDOWN, VHDOWN(the methods used to define the state are discussed later) . These categories are formed from a day trading session based on the percent change between the open and close price of the stock. If the close price is higher than the open price it is a positive price movement and we call this state UP, if greater than 1% it is HUP and greater than 2% it is VHUP. Similar logic is used to define the other 3 states. This method will be especially useful in case of computational constraints. Finally, we defined the actions a trading agent can perform(also expressed below).The trading agent in our project is implemented using the off -policy Q learning method RL. The algorithm maintains a table of all state-action value pairs(called a Q-table) which it will update for every Q(s,a) action based on the algorithm and formula:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

To ensure the algorithm explores possible states we implemented ϵ-greedy Q learning algorithm with the value of ϵ being 1 initially(full explore) and as the Q-table converges, ϵ decreases.

A FOREWORD ON SOME OF THE CONCEPTS USED IN THE PAPER:

1) DEFINING THE ACTIONS

Before we dive into the implementation, we must firstly define the basic fundamentals used. In a real-life scenario, day trading can only be done in the form of **Margin Trading -** a form of trading wherein the trader takes certain loans/margins from a brokerage firm to generate capital for trading and on a day-to-day basis evaluation of his portfolio(margin) is done to see how much profit he has. The trader will always have to be able to repay the brokerage at any point of time and must be able to pay back arrears when a **margin call** is performed -else he risks losing his

portfolio. Also, the brokerage charges a brokerage fee for every day that trader uses the margin which is also calculated based on an industry rate. We have implemented this real-life scenario by clearly defining certain variables that will be used to change the day end capital trading agent will have based on the actions taken.
Also, one must understand two more key terms - a short position and a long position. A **short position** is one wherein the trader bets the share price will go down in the future and cashes in when it does. This is implemented by borrowing the stocks from the brokerage firm today and selling it back to them if and when the price goes down in the future. You will get to keep the change and this strategy helps traders make money even when the market is down. Example : You borrow 100 stocks of a particular company from a brokerage firm and then sell it at a current price(say 10$). When the price goes down to 5$ in the future -you buy those stocks back and return the 100 stocks to the brokerage. That way you get to keep the change of 500$.
A long position is diametrically the opposite of this- you bet the stock price will go up and cash in when it does - so you buy now and sell later at a high price.

Now that this has been understood, we shall proceed in explaining how exactly actions have been defined and how subsequent rewards for taking the particular action have been computed.
We defined 3 actions and they basically represent the POSITION trader must follow if and when this action happens:
ACTION =0: This action basically translates to follow whatever position you were following in the time step before the current time step. This means that if we were following a long position , the agent must keep at it and similarly for a short position. However, if the loss made by the trader by following the position is greater than 10%(w.r.t closing price at that time step), then under this action -Trader must stop following the policy - meaning CLOSE THE POSITION. The algorithm will understand that if the previous sub action taken by the agent was to buy the stock to be that agent is following the long position. Similarly, if the sub-action(as explained in the code) taken by the agent was to sell the stock, the algorithm inferred that the agent was following a long position.
ACTION=1: This action translates to CLOSE A SHORT POSITION if the agent had been following that position and making a loss of more than 10% at the time step CONTINUE WITH A LONG POSITION if it was following a long position irrespective of whether the agent was making loss or profit then. If an agent hasn't followed a position before this meaning this is the first action it takes, then we initialize the agent to START WITH A LONG POSITION. The algorithm will know which position the agent was taking based on a sub-action routine we defined which takes 3 values -BUY,SELL OR NONE. If sub-action routine ==SELL -It was following a short position and if sub-action routine==BUY it was following a long position.
ACTION=2: This action translates to CLOSE A LONG POSITION if the agent had been following that position and making a loss of more than 10% at the time step and CONTINUE WITH A SHORT POSITION if it was following a short position irrespective of agent making loss or profit then. If an agent hasn't followed a position before this meaning this is the first action it takes, then we initialize the agent to START WITH A SHORT POSITION.
While the two models differ in the way we defined the states - the action taken is the same for both and is based on the above logic which we have implemented in our code.
As a result, the action which the agent can take at any state is from a tuple of size 3. We also defined the selection of action in such a way that based on the ϵ of the agent in that particular episode ,we take random action ϵ number of times and the rest 1- ϵ we take greedy action.

2) DEFINING THE REWARD

The reward definitions were done based on some of the criteria which must be satisfied based on the profit agent makes at that time instant, the margin/ portfolio value being managed and finally

the profit percentage. By default if the agent's action taken doesn't satisfy any of the criteria mentioned below, that time step the agent has 0 reward.

The conditions are:

1)If in a particular time step in the episode, the agent does not change the position it was following previously(meaning it is better to wait to sell if you are going long and wait to buy knowing price will go down if you are shorting), the reward will remain 0 for that iteration of the Q learning algorithm.

2) If at a particular time step we see the profit made from selling a stock while following action 1 is greater than 10%(outlier -sure that stock price will only go up from here and no more gains to be made by shorting - best to close your short now) or if the loss was more than 10%(meaning it is a poor decision to short(sell) at the current time step- we should buy back the shares in the previous time step else risk making a loss) ,we reset the sub-action routine as NONE and update the reward as the maximum among 0 and the difference of the Logarithmic price of closing stock at current time step and the price at which we started shorting stock. Also, the value of profit(net gains) is updated as the differences in price of the stock from the time we started shorting to the time we ended it(buying back) times the quantity bought at the time of shorting. The margin is also updated accordingly.

3) If at a particular time step we see the profit made from buying a stock while following action 2 is greater than 10%(outlier -sure that stock price will only go down from here and no more gains to be made by going long- best to close long position now) or if the loss was more than 10%(meaning it is a poor decision to long(buy) at the current time step- we should have sold our shares at the last time step else risk losing profit) ,we reset the sub-action routine as NONE and update the reward as the maximum among 0 and the difference of the Logarithmic price of closing stock at current time step and the price at which we started shorting stock. This quantity is 0 if we were making a loss following the previous position now.

The way we define the profit variable ensures that we only update the profit whenever a position shift happens and we only update the last price whenever we are shifting our policy - this helps us maximize our profit as much as possible.

Now that we have explained the meaning of our actions in calculating profit with each time step and also explained how rewards are computed, we shall move to implementing the models.

SOME ADDITIONAL INFORMATION ON HOW ACTIONS ARE CHOSEN IN TESTING :

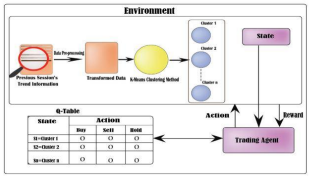
In the algorithm which we have implemented, we try to actually improve the action taken by the agent. We created a function that has an array data type that is initially a null array of size 3 which will be updated based on some conditions and this updated list is returned by this function. We brought in another parameter around consideration- **Relative Strength index.** The relative strength index (RSI) is a momentum indicator used in technical analysis that measures the magnitude of recent price changes to evaluate overbought or oversold conditions in the price of a stock or other asset.Traditional interpretation and usage of the RSI are that values of 70 or above indicate that a security is becoming overbought or overvalued and may be primed for a trend reversal or corrective pullback in price. An RSI reading of 30 or below indicates an oversold or undervalued condition. We compute if a stock is an overbought or oversold position based on the closing price and for each day of the trading information given-this must be calculated.

While testing, we checked the value of the RSI for that time step for each episode and if the value was greater than 70 we will make ls[0]=2 to indicate that it would be better to sell(the meaning of 0 1 and 2 are elucidated in the above code). If in between 70 and 30 we say continue with position i.e ls[0]=0. And if less than 30, we say best is to change position to sell i.e ls[0]=1.

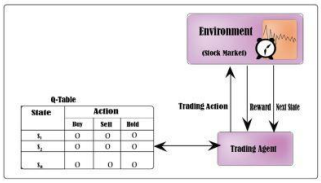
Next,we discussed a term **Williams Percent Range** which is another indicator to indicate if the market is overbought or oversold(based on data over a 14 day window) and again based on it's values we updated the 2nd index of ls (i.e ls[1]) and finally we had another term **Commodity Channel Index** which is another indicator to indicate overbought or oversold levels (which is updated into ls[2]).

The whole point of doing so is to ensure a position change is taken up by the agent if and only if the conditions of the market are such that it would pay to change the position.
For example, if the agent is following ACTION==1 for this time step(which indicates to continue with a long position and close short positions) and suppose at that instance the value of the indicator variable is 1 -it would mean the stock is in an oversold condition. If the last sub-action routine taken by the agent was to sell we correct it's action(CLOSE SHORT) if and only if the market is so inflated that agent makes loss of more than 5% or a profit of more than 10% following a short position-meaning selling is grossly overvalued and most likely this position is suboptimal for the long run.
Similarly, if the agent is following ACTION==2 for this time step(which indicates to continue with a short position and close long positions) and suppose at that instance the value of the indicator variable is 2 -it would mean the stock is in an overbought condition. If the last sub-action routine taken by the agent was to buy we correct it's action(CLOSE LONG) if and only if the market is so inflated that the agent makes a loss of more than 5% or a profit of more than 10% following a long position-meaning buying is grossly overvalued and most likely this position is suboptimal for the long run.

The benefit of adding another indicator to the action decision tree(which in turn influences the reward inputted to the Q-learning algorithm) is MAXIMIZING PROFIT FROM POSITION FOLLOWED BY THE AGENT. In testing, we have a single episode running over the testing period and we do not have the luxury to follow a suboptimal position that is holding back our profit possibilities. This ensures that while the best actions from the Q-table are being selected, we are also reducing any loss when a random action is being selected as well. The contents are further understood in the code written in the collab/jupyter notebooks..



MODEL 1 VISUALIZATION



MODEL 2 VISUALIZATION

IMPLEMENTATION OF METHOD 1 : CLUSTER TO REPRESENT A STATE

In this model, we use the previous data and keep updating our model based on how the model learns from past experiences. We can assume that history has repeated in this model i.e. we can assume that the current trading pattern can be similar to the trading pattern which has happened in the past. In any pattern of trading, three things can happen which are uptrend, downtrend and side trend. These trends depend on the initial and final position of the stock value. The main goal of this model is to find a past trading session that can give the most optimal action and use it to get an optimal action in the present situation. For this model we use KMeans Clustering algorithm along with reinforcement learning to develop the model.

Data:

The data used for training is the DJIA data set which has been obtained from Yahoo Finance. The data contains 4577 rows and columns contain the following:

1) Date: Date of the stock
2) High: Highest price of the stock for that date
3) Low: Lowest price of the stock for that date
4) Open: The open price of the stock for that date
5) Close: The close price of the stock for that date
6) Volume: Amount of stocks that have been sold or bought that on that date
7) Adjusted Close (Adj Close) : This is the price which changes the close price of the stock to reflect the stock's value after taking into consideration any corporate actions.

This is how the first five rows of the data looks like:

	Date	High	Low	Open	Close	Volume	Adj Close
0	2000-10-19	10142.980469	10014.610352	10014.610352	10142.980469	343710000	10142.980469
1	2000-10-20	10229.549805	10067.509766	10141.129883	10226.589844	319750000	10226.589844
2	2000-10-23	10361.250000	10216.230469	10230.290039	10271.719727	339180000	10271.719727
3	2000-10-24	10439.309570	10273.570312	10273.570312	10393.070312	274990000	10393.070312
4	2000-10-25	10461.969727	10306.589844	10395.660156	10326.480469	312490000	10326.480469

This is more information about the data types of each column:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4577 entries, 0 to 4576
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       4577 non-null   object 
 1   High       4577 non-null   float64
 2   Low        4577 non-null   float64
 3   Open       4577 non-null   float64
 4   Close      4577 non-null   float64
 5   Volume     4577 non-null   int64  
 6   Adj Close  4577 non-null   float64
dtypes: float64(5), int64(1), object(1)
memory usage: 250.4+ KB
```

The columns High, Low, Open and Close have been preprocessed to form columns O,H,L,C,V. These columns represent the state of the environment. Below the formulae of the mentioned columns are given:

1) O= Percentage of change in the open price as compared to the close price of the previous day.
   O= (Open(t)-Close(t-1))/(Close(t-1))*100
2) H = Percentage change in high price as compare to same day open price
   H= (High(t)-Open(t))/(Open(t))*100
3) L= Percentage change in low price as compared to same day open price.
   L= (Low(t)-Open(t))/(Open(t))*100
4) C= Percentage change in close price as compared to open price of the same day.
   C= (Close(t)-Open(t))/(Open(t))*100
5) V= Percentage change in volume as compared to the previous day volume
   V= (Volume(t)-Volume(t-1))/(Volume(t-1))*100

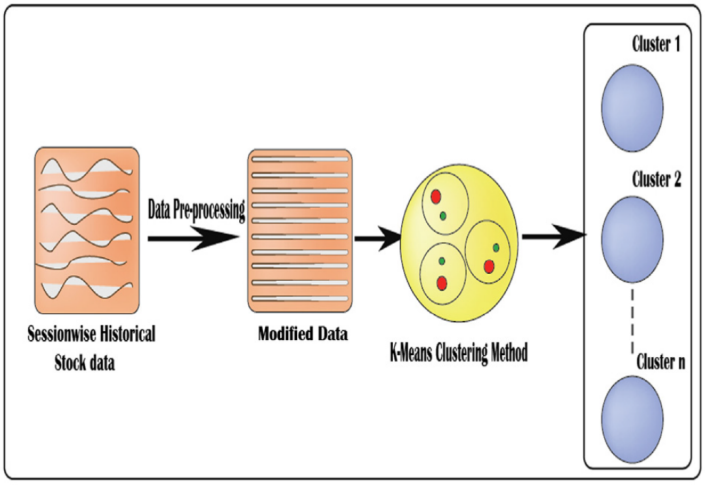
Three more columns namely Cash, Action, Action1 have also been added in the data set which are defined below:

1) Cash= The amount of money we have at the end of a day of trading.
2) Action= Close short, Close long, Open short, Open long
3) Action1= Short, Long

Model Implementation:

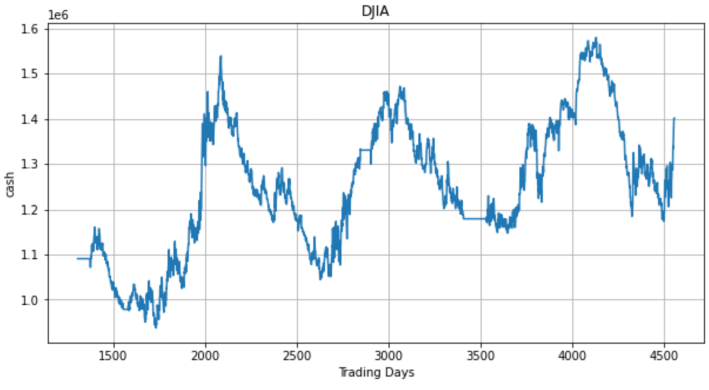
The data which has been processed is fed in the K Means Clustering algorithm. The algorithm forms n clusters where n is a hyperparameter. The agent used for trading is implemented using the Q learning algorithm of reinforcement learning. This algorithm contains a Q table whose values are updated after each episode to get an optimal trading algorithm. There are n states in the Q table and the actions are buy, sell and hold depending on the present action and the last action performed. These can be further classified into close short, close long, open short and open long. The Q-value for each state action pair of Q-table is updated based on the rewards being obtained after each episode. Here episode refers to one iteration of the entire data set.

The data is divided into training and testing where the first 1304 entries are taken for training and other 4557-1304=3253 rows are taken for testing. Below is how clusters are formed from the data in the stock market which has been used in this model:



(Source: A Q-learning agent for automated trading in equity stock markets Jagdish Bhagwan Chakole, Mugdha S. Kolhe, Grishma D. Mahapurush, Anushka Yadav,Manish P. Kurhekar)

Results and Interpretations of Model 1:



The side graph explains the change of the available capital (Which includes the profits) w.r.t. the time. We can observe a sideways trend within them.

```
Episode 1/1
--------------------------------

**********************
Profit is  310455.6591826172
Initial Capital is  1090849.0234375
Current Capital is  1401304.6826201174
%ROI is  28.460002485431474
% Buy and Hold= 112.28431473210006
count 12
qtable [[1.68954948e-06 5.23218001e-03 1.66540424e-06]
 [3.38525589e-04 0.00000000e+00 0.00000000e+00]
 [2.54924398e-04 3.98425926e-05 8.58331412e-05]]
epsilon 0.00998645168764533
--------------------------------
e=1,roi=28.460002485431474
Maximum profit is -1000000
Total profit is 0
```

When we run the developed algorithm on the testing data for one episode. We ended up with a profit around **3,10,455** (as per the latest run) And rest of the stats can be found through the provided snippet.

IMPLEMENTATION OF METHOD 2: DEFINING STATES USING CANDLESTICK DATA

HOW THE STATES ARE DEFINED AND COMPUTED:

As mentioned earlier, we defined 6 states in the following method. The states are calculated using the formula :

$$\frac{(CLOSING\ PRICE - OPEN\ PRICE)}{OPEN\ PRICE} \times 100\%$$

Where closing price and opening price are available from the stock data which we try to analyze.
The states are classified according the percentages obtained as listed in the table given below:

State representation for the proposed model 2 using candlestick.

%[(Close-Open)/Open]	State
(0,1)	UP
(1,2)	HUP
(> 2)	VHUP
(−1,0)	DOWN
(−2,−1)	HDOWN
(< −2)	VHDOWN

To implement, we assigned numerical values to each state starting from 0 to denote VHDOWN and finally 5 to denote VHUP.
This column data array was appended to our data frame that we imported to represent the particular stock.
As you can already see, this model tries to reduce the computational requirements for the algorithm while at the same time leaving the states less-exhaustively defined.

We then defined a class to define the agent(with constructors to denote it;s epsilon value and epsilon decay rate) and defined sub-modules to describe what it must do to update q values(using

update**(self,state,action,reward,next_state,next_action)**) and what it must do to generate a new state based on our epsilon greedy approach(using **getAction(self,state)).**

For implementing the Q-learning algorithm, we split the data given about the stock into Training and Testing data periods(we train the model for the first 1305 days of data and test the algorithm for the rest of the days). And based on how we defined our states and rewards(as mentioned above) we implemented the training and testing algorithm.
In the algorithm, we printed what profits the agent makes for that particular episode as it undergoes training and at the end of each episode we also printed the return of investment to denote how well our algorithm was able to implement the automation. We ran the above code for 200 different episodes.
For the purpose of this report, we ran the training and testing models for various types of stocks but to reduce the computational burden we used 1 index stock and 2 individual stocks.
The index stock which we used for our purpose is DJIA - which is an aggregate of the stock prices of the top rated companies/enterprises in the US.
The results of our implementation of Model 2 has been listed below:

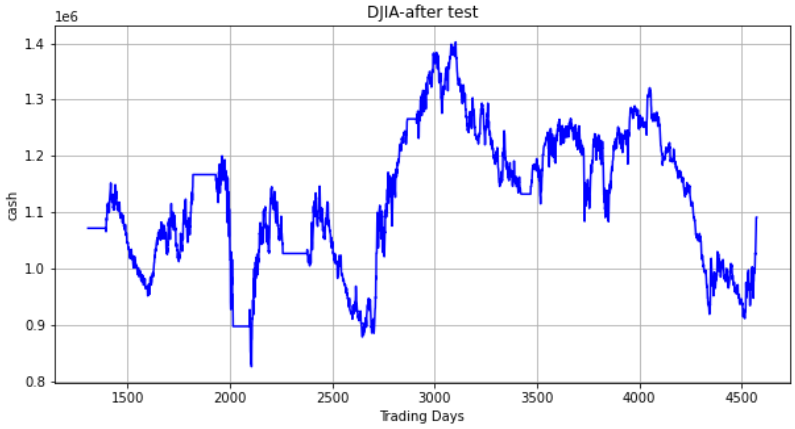
RESULTS OF MODEL 2

As mentioned above, this algorithm is an ϵ greedy Q learning algorithm- so we select random action with epsilon probability at each time step and 1- ϵ we choose the action giving the highest value of Q(s,a) for the particular candlestick state we are in. We ensured that epsilon decays with each time step by a factor (shown in source code) that ensure towards the end of each episode - we would be choosing the action giving highest Q(s,a) from the Q-table - meaning we reduce exploration of states as we reach optimal solution/converge to it.
Let's start with showing the Index Stock(DJIA) and it's closing price graph over the days:

We shall now show how our model's training data helped our agent get some form of capital gains:

For the testing block, we basically took the Q-value table which was obtained after running the training model for 200 episodes - where in each episode we started with epsilon=1 and we kept decaying it as the

episode progressed but we used the same Q-table to select our greedy actions and the same Q-table was being re-fitted every episode.
This Q-table values along with the final epsilon value(which would be about initial_value*decay_rate**(no.of steps in episode)) ,since we are now interested in using the optimal Q-table actions only for each state and we don't want to bother exploring for the testing case, was inputted into the testing algorithm as the agent's initial Q-table and initial Epsilon at the start of the episode. Based on the optimal Q-table obtained after training, we will deploy our algorithm to see how well it will work.
Using the Q-value table, the following shows how for the rest of the days of data collected for DJIA stock, our capital was changed:



The following image shows the net capital change we made from our initial capital(day(0)(opening_price)*100) to the final day and the profits made:

```
Episod 1/1
--------------------------------

**********************
Profit is  19346.73721582032
Initial Capital is  1071829.98046875
Current Capital is  1091176.7176845705
%ROI is  1.8050192258439572
% Buy and Hold= 113.31193325595792
count 18
qtable [[0.00165828 0.00730093 0.00147067]
 [0.00073959 0.00073521 0.00077549]
 [0.00035929 0.00028909 0.00028723]
 [0.00026842 0.0004264  0.00026699]
 [0.00065044 0.00066163 0.00065195]
 [0.00140939 0.00216879 0.0035898 ]]
epsilon 0.00998645168764533
--------------------------------
e=1,roi=1.8050192258439572
```

So, we can see that after running our model - we have made a small profit from trading by using Model 2.

It can be seen that this model greatly outperforms the Buy and Hold Policy the agent would have taken(which is basically to buy as many stocks as you could initially based on the capital you have(in this case day 1306 of data collection) and sell it on the last day).

IMPROVEMENT TO OUR ALGORITHM - DOUBLE Q LEARNING

Inspiration for the improvement:

In the algorithm, the rewards which are being input into the algorithm is the difference between the logarithm of the closing price at the time step and price at which stock was bought or sold and 0 -whichever one is highest based on the conditions discussed above.

If one were to keep 2 Q-tables and update one based on the maximum state-action pair for corresponding state from the other , we are able to **converge faster onto the optimal policy** . This basically means that we will be able to reach the best position the agent should take quicker since we will be able to overcome bias that would have developed from a certain series of actions taken since with equi probability we can either update the values of table 1 or table 2 when the action is a greedy action. It is able to **overcome maximization bias** - so that if we were consistently choosing one action n times, in the current scenario we would be doing so n/2 times. Since, our actions weren't as well defined to account for all contingencies - we would definitely have seen the aforementioned maximization bias which is being partially offset using double Q learning.

So now, we will converge faster onto the optimal policy agent must take and as a result we would be able to improve the capital gains at the end of the testing period thereby increasing profit the agent would have made. We shall now demonstrate how we will get better results .

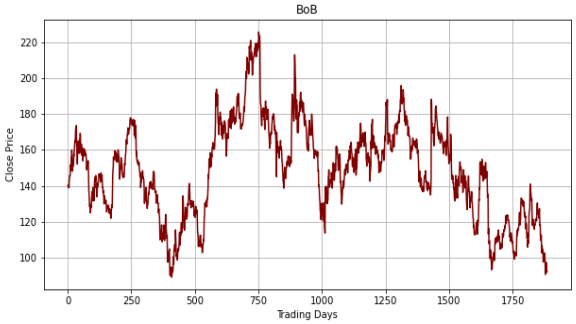
Implementation:

We just made 2 subtle changes while implementing this algorithm. Firstly, we defined two Q-table matrices(and since we are only showing improvements for model 2 due to computation constraints) of size 6 x 3 within the constructor that defines the agent. In the action generating function:
While epsilon times we will select random actions, of the 1-epsilon times we select greedy action, 0.5 times we select greedy action from Q-table 1 and the rest 0.5 times we select greedy action from Q-table 2.
Also, while updating the value of the Q-tables again with a probability of 0.5 we update the value of Q-table 1 for the given state based on the maximum value of the corresponding state's (s,a) pair in the Q-table 2. And the rest of the time, we do the vice-versa.

COMPARISON B/W THE OUTPUT OF MODEL 2 AND IMPROVED ALGORITHM (DOUBLE Q LEARNING) FOR BANK OF BARODA STOCK

Model 2 result for Bank of Baroda Stock:

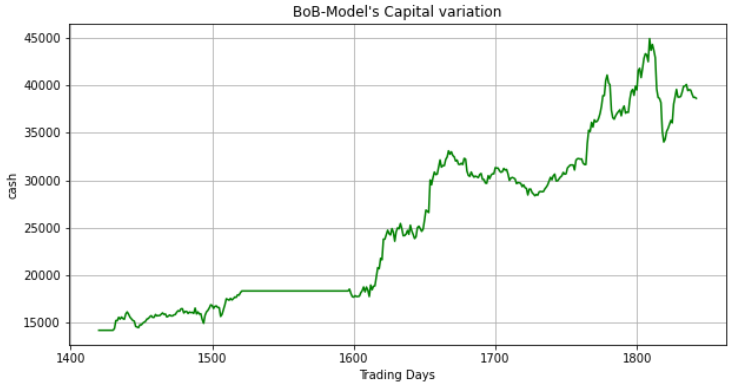
We have taken stock data of **BARODA BANK** over the period from 2012-01-01 To 2019-08-31. We shall test how well our algorithm works for an individual stock. We trained our model from 2012-01-01 to 2017-09-28 and the rest of the days we tested our algorithm.
Let's start with showing the Individual Stock(BoB) and it's closing price graph over the days:



We shall now show how our model's training data helped our agent get some form of capital gains:

At the end of training, we see that the capital agent will have at the end of the 2000th episode(in each episode everything except the Q-table is the same -Q-table is what is being updated at each episode) the agent has made poor decisions which has resulted in less Capital gains.

Based on the Q-table obtained after Training let us make the model run for the remaining trading days in a single episode to see just how successful it actually is in generating profit from the initial capital we have(which in this case will be assumed to be the open price of stock on the1420th day*100).
The following is the result obtained:



We see that contrary to our understanding based on the training result, the agent has been very efficient and has devised a position across the testing days that seems to have generated a humongous amount of profit.

The final table to denote the results from our episode:

```
Episod 1/1
--------------------------------

**********************
Profit is  31264.744811264034
Initial Capital is  14189.999389648438
Current Capital is  45454.74420091247
%ROI is  220.32943027518078
% Buy and Hold= -14.154603802535643
count 7
qtable [[0.00480624 0.01104496 0.00506591]
 [0.00429586 0.00483941 0.00427751]
 [0.00456517 0.00374874 0.00374443]
 [0.00492877 0.00437352 0.00440027]
 [0.00520645 0.00379729 0.0037714 ]
 [0.00414702 0.00449949 0.01983234]]
epsilon 0.00998645168764533
--------------------------------
e=1,roi=220.32943027518078
```

While the agent has generated a good amount of profit based on his initial capital- we see one major blow here - the profit made is about 14% of what the agent would have made had he just followed a Buy and Hold Position -meaning he bought up shares at the starting day based on his capital he had, did nothing for the whole testing period and sold the stock on the last day.
This tells us two things:

1) The number of days for which the sample of stock prices were collected is inadequate for the model to train and then test it's best policy - if we had a sample of greater number of days we could have had a greater variation and the agent would definitely have been able to beat Buy and Hold.
2) The stock has been in an Upward Trend over the whole testing and training period-hence such a skewed result.

Output using the IMPROVED DOUBLE Q LEARNING MODEL

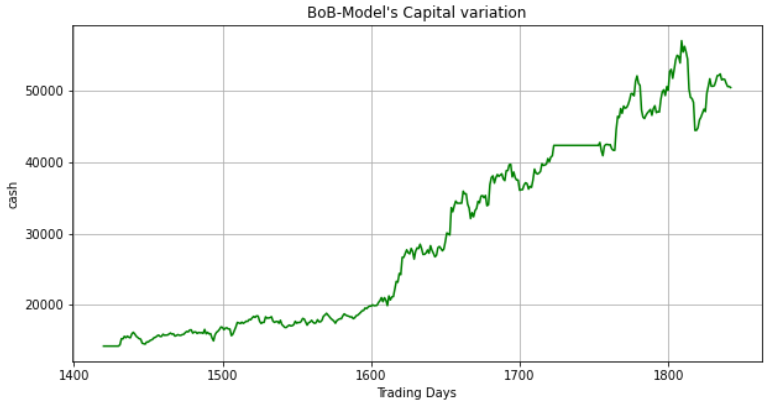
We shall firstly see how well our agent has been trained by seeing the plot of it's capital gains made in the final(200th episode) based on the actions it has taken from both the Q-tables defined:

Now based on the new algorithm used, again we see the agent has made poor decisions which has resulted in it seeing a fall in capital for the training days.

After the training period, we have obtained a tuple of two converged Q-tables. For the testing purposes in the remaining time steps- we use these two Q-tables to generate the actions the agent must take for the testing period. We will also import the value of epsilon from the 200th episode

(which would be very small) to indicate that while the algorithm used is still an epsilon greedy one- we do not require the exploratory actions anymore as much since we have generated an optimal understanding of which action will give us best reward.

The following is the result of the agent's capital after the testing period(from day 1420 to day 1864) for a single episode.



The final table to denote the value of profit and the likes is also given below:

```
Episod 1/1
--------------------------------

**********************
Profit is  36302.42386937713
Initial Capital is  14189.999389648438
Current Capital is  50492.42325902557
%ROI is  255.83104602428412
% Buy and Hold= -14.154603802535643
count 7
qtable1 [[0.00564446 0.00670422 0.00581756]
 [0.0058717  0.01503336 0.00541911]
 [0.00531753 0.0062172  0.00546568]
 [0.00489557 0.0053795  0.00515829]
 [0.00539146 0.00554527 0.0101847 ]
 [0.00451931 0.00465744 0.00920006]]
qtable2 [[0.00519549 0.00533584 0.00521596]
 [0.00532665 0.01215693 0.00519221]
 [0.00475262 0.00554906 0.00511912]
 [0.00462831 0.00517617 0.00605584]
 [0.00469921 0.00540657 0.00950186]
 [0.00431472 0.00556518 0.02025404]]
epsilon 0.00998645168764533
--------------------------------
e=1,roi=255.83104602428412
```

Comparing the profit we obtained with Model 2 without Q-leaning we see the following results:

1) We have been able to increase the profit the agent generated again demonstrating how double Q-learning is able to converge faster to the optimal solution.
2) The ROI % has also increased for the double Q-learning model which again shows that we are able to get better results.

Hence, DOUBLE Q-LEARNING HAS ENABLED US TO IMPROVE THE PROFIT MODEL 2 WOULD MAKE IN COMPARISON TO USING ONLY A SINGLE Q-TABLE GIVEN THE SAME DATA SET AND AS A RESULT, OUR MOTIVATION FOR SUCH AN IMPROVEMENT IS JUSTIFIED.

CONCLUSION

We have been able to implement our algorithm that helps the agent perform automated stock trading by using Q-learning to make the algorithm choose which action will give the maximum profit. We were able to do so by using two models to represent the states for the agent - In the first model we used K-means clustering(which in this case was 3) tp represent the states which had some similarities between them and in the second model we used CANDLESTICK data to represent 6 states based on the share price.

We then performed our RL algorithm using both model 1 and model 2 to make the agent trade stocks and maximize profit based on the initial capital and to compare their performance, we used a common stock index DJIA to see the effects of our results. The following were our results:
1)While we generated a net profit of about 32,000 using model 1 - model 2 was only able to generate 20,000 dollars of profit. Hence, **our model 1 was better than model 2.**
2) **Both models greatly outperformed the Buy and Hold policy** implying the agent in both models made smart decisions to increase profit.

We then tried to improve our Q-learning algorithm and we **used double Q-learning to help the agent converge faster to optimal action and minimize maximization bias** and on comparing the results of the improved model, the improved model was able to outperform the model 2.

CONTRIBUTIONS :

NAME	ID Number	CONTRIBUTION
JEEVAN REJI	2019AAPS0297H	● Explanation of actions taken by agents and how rewards are computed. ● Model 2 Implementation ● Improvement of existing algorithm
AVINASH GONDELA	2019A8PS1357H	● Model 1 Implementation ● Improvement of existing algorithm
VANAMA PHANINDHAR	2019AAPS0253H	● Model 1 Implementation ● Improvement of existing algorithm
BHARAT K	2019A7PS0098H	● Web scraping for stock data(DJIA) ● Improvement of existing algorithm
SAI RAGHURAM	2019A8PS0528H	● Web scraping for stock data(BoB) ● Improvement of existing algorithm