CSE 574 Introduction to Machine Learning

Project 3

Rahul Reddy Karumuri (50416790)

1. Overview:

The goal of this project is to implement Q Learning Algorithm from scratch on Nvidia dataset and to implement a trading strategy to maximize the profit. In each trade we can either buy/sell/hold. We will start with an investment capital of $100,000 and the performance is measured as a percentage of the return on investment. We have to use the Q-Learning algorithm for reinforcement learning to train an agent to learn the trends in stock price and perform a series of trades.

2. Dataset:

Dataset that we need to consider is Nvidia dataset. This dataset contains 1258 records of stock market data for the last 5 years starting from 10/27/2016 to 10/26/2021 which includes the information such as the price at which stock was opened, and the intraday high and low, and the price at which the stock has closed, the adjusted closing price and the volume of shares traded for the day. Here is the sample the dataset

NVDA

Date	Open	High	Low	Close	Adj Close	Volume
2016-10-27	18.177500	18.212500	17.597500	17.670000	17.416187	38866400
2016-10-28	17.754999	18.025000	17.607500	17.639999	17.386621	29085600
2016-10-31	17.697500	17.907499	17.687500	17.790001	17.534472	25238800
2016-11-01	17.855000	17.952499	17.072500	17.262501	17.014545	47322400
2016-11-02	17.395000	17.629999	17.160000	17.190001	16.943085	29584800
2016-11-03	17.270000	17.285000	16.660000	16.990000	16.745956	30966400
2016-11-04	16.877501	17.182501	16.645000	16.892500	16.649853	32878000
2016-11-07	17.387501	17.930000	17.375000	17.817499	17.561563	48758000
2016-11-08	17.885000	17.942499	17.625000	17.790001	17.534472	42988400
2016-11-09	17.307501	17.725000	17.180000	17.490000	17.238771	45653200
2016-11-10	17.872499	17.875000	16.690001	16.942499	16.699137	86928000
2016-11-11	19.877501	22.192499	19.625000	21.992500	21.676601	217534400
2016-11-14	22.022499	22.047501	20.905001	20.910000	20.609653	134879600
2016-11-15	21.072500	21.862499	20.982500	21.547501	21.237995	62609200
2016-11-16	21.834999	23.139999	21.587500	22.907499	22.578461	98798400
2016-11-17	23.077499	23.697500	22.662500	23.097500	22.765728	83298800
2016-11-18	23.097500	23.582500	22.950001	23.340000	23.004744	57606000
2016-11-21	23.522499	23.587500	23.100000	23.245001	22.911106	43643600
2016-11-22	23.325001	23.434999	23.094999	23.412500	23.076204	33036800
2016-11-23	23.430000	23.812500	23.192499	23.492500	23.189724	44323600

3. Environment:

I have used Jupiter Notebook on Google Colab for implementation of this project.

4. Implementing Q Learning Algorithm:

Q Learning is a type of Reinforcement Learning technique that enables an agent to learn the environment by trial and error method and get feedback from the environment from its own experiences

4.1: Environment Structure

Environment: Physical world in which the agent operates
State: Current situation of the agent
Reward: Feedback from the environment

In our project Environment was already given, we have to implement the Q Learning model for the given Environment

Methods in environment:

Init method: This method initializes the environment.

Input parameters:

1. file_path: Path of the CSV file containing the historical stock data.
2. train: - Boolean indicating whether the goal is to train or test the performance of the agent.
3. number_of_days_to_consider = Integer representing whether the number of days the for which the agent considers the trend in stock price to make a decision

Reset: The reset function returns the current observation in the range from 0 to 3
Where each integer indicates

Observation [0,1,2,3]:

Where,
0. Price of the stock has increased, and the agent doesn't hold any shares.
1. Price of the stock has increased, and the agent holds shares.
2. Price of the stock has decreased, and the agent doesn't hold any shares.
3. Price of the stock has decreased, and the agent holds shares.

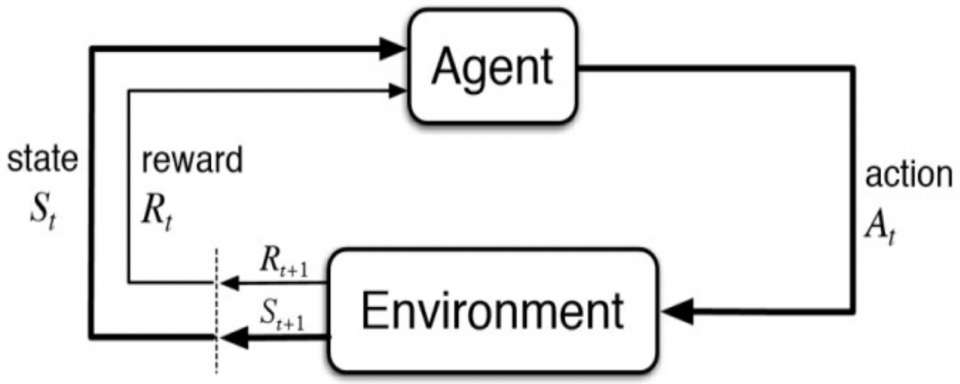
Step method: This method implements what happens when the agent takes an action to either Buy Sell or Hold the stock. Input parameter: action: - Integer in the range 0 to 2 inclusive. Where,

Actions[0,1,2]:
Where,
0 means buy the stock
1 means sell the stock
2 means hold the stock

Returns:

1. observation
2. reward: - Integer/Float value that's used to measure the performance of the agent.
3. done: - Boolean describing whether or not the episode has ended.
4. info: - provides information on additional implementation.

Render method: This method renders the agent's total account value over time. Input parameter: environment



4.2 Algorithm for Q Learning:

1. Initialize the hyperparameters (alpha, gamma, epsilon), Q Table to zeros (Q Table should be 4x3 matrix since there are 4 states and 3 actions)
    2. Iterate through each episode
        2.1 Reset the environment for each episode
    2.2 Iterate through each step-in episode
        2.3.1 Take a random value if the random value is greater than epsilon then exploit else explore
        2.3.2 Call the step function in environment to take an action (buy, sell, or hold)
        2.3.3 Now update the Q Table with the below formulae

$$Q(s',a) \leftarrow (1-\alpha)\cdot Q(s,a) + \alpha \cdot (r + \gamma \cdot \max_{a'} Q(s',a'))$$

        2.3.4 Update the state to new state
        2.3.5 If the loop is done then break it

Methods I used for implementing Q Learning algorithm:

Init method: This method initializes the variables required for solving the algorithm and takes the environment as the parameter

Variables I had used:
actionCount : Action size = 3 (buy, sell or hold)
stateCount : Observation size = 4 (0,1,2,3)
QTable : Matrix of size stateCount x actionCount
Episodes : Total number of episodes for traning the model
maxIterations : Maximum number of iterations per each episode
alpha : Learning Rate
gamma : Discount factor
epsilon : exploration rate
minEpsilon: minimum exploration rate such that when decaying the epsilon it shouldn't cross below this value
episodesArray : for plotting the graphs
rewardArray : for plotting the graphs
epsilonArray : for plotting the graphs

Train Method: Here is the function I used to train the agent based on the above algorithm

```
def train(self):
    """This method performs the agent training."""

    #Training the Agent
    for episode in range(self.episodes):
      #Reset the environment for every iteration
      state = self.environment.reset()
      print("**********      Episode:",episode,"      *********")
      #This loop will breaks if it is done or it reach the max iterations of 1000
      for iteration in range(self.maxIterations):
        #Exploitation
        if np.random.uniform(0,1) > self.epsilon:
          action = np.argmax(self.QTable[state,:])
        else: #Exploration
          action = self.environment.action_space.sample()

        #Call the environment step function and store the returned values
        newState, rewards , done, info = self.environment.step(action)
        #Updating the Q Table
        #Formulae to calculate the Q Value
        self.QTable[state, action] = (1 - self.alpha) * self.QTable[state, action] + self.alpha * (rewards + self.gamma * max(self.QTable[newState]) - self.QTable[state, action])
        #Adding the reward to the total rewards for plotting the graph
        self.rewardArray[episode] += rewards
        #Assigning the newState to the state, so used for the next iteration
        state = newState
        #Breaking the for loop if the episode has ended
        if done:
           break

      #Reducing the epsilon value, since we need less exploration in every step
      if self.epsilon > self.minEpsilon:
        self.epsilon *= 0.995
      #Storing the Epsilon values in an Arrray
      self.epsilonArray.append(self.epsilon)

    #printing the Q Table
    print("Q Table:")
    print(self.QTable)
```

Evaluate: In the evaluate function I am testing the trained agent and calculated the total reward after training. Here is the function I used for evaluating the trained Agent

```
def evaluate(self):
    """This method evaluate the trained agent's performance."""

    #Flag to enable evaluate
    self.environment.train = False
    #Reset the environment
    state = self.environment.reset()
    #This loop will breaks if it is done or it reach the max iterations of 1000
    for iteration in range(self.maxIterations):
        #exploitation
        action = np.argmax(self.QTable[state,:])
        #Call the environment step function and store the returned values
        newState, rewards , done, info = self.environment.step(action)
        #Formulae to calculate the Q Value
        self.QTable[state, action] = (1 - self.alpha) * self.QTable[state, action] + self.alpha * (rewards + self.gamma * max(self.QTable[newState]) - self.QTable[state, action])
        #Assigning the newState to the state, so used for the next iteration
        state = newState
        #Breaking the for loop if the episode has ended
        if done:
          break
```

Plot: This function is used for plotting the results

i.e. 1. Episodes vs Reward Dynamics

2. Episodes vs Epsilon Decay

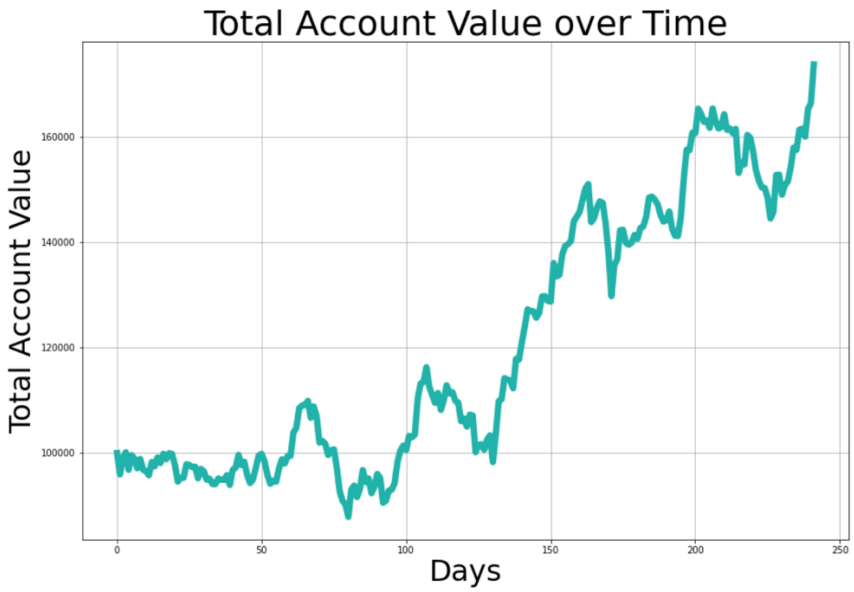
Here is the function I used for plotting the graphs

```
    def plot(self):
        """This method plots the reward dynamics and epsilon decay."""

        #Plotting the Rewards Dynamics with episodes
        plt.figure(figsize=(11, 7))
        plt.plot(self.episodesArray, self.rewardArray, color='blue', linewidth=2)
        plt.xlabel("Episodes",fontsize=24)
        plt.ylabel("Rewards values",fontsize=24)
        plt.title('Total rewards per episode', fontsize=32)

        print()
        print()
        print()

        #Plotting the Epsilon Decay with episodes
        plt.figure(figsize=(11, 7))
        plt.plot(self.episodesArray, self.epsilonArray, color='red', linewidth=5)
        plt.xlabel('Episodes', fontsize=24)
        plt.ylabel('Epsilon values', fontsize=24)
        plt.title('Epsilon Decay', fontsize=32)
        plt.show()
```

4.5 Results:

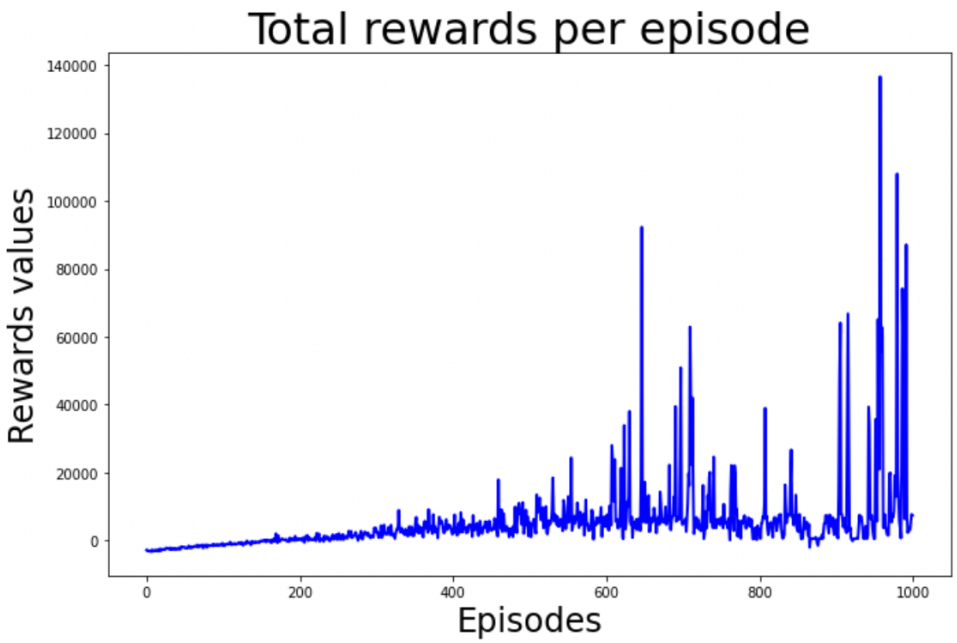
Total Account Value after Evaluating the above model: 1,73,752.2127

```
print()
QLearning.evaluate()
stock_trading_environment.render(QLearning)
#Printing the Total Account Value after evaluating
print("Total Account value after Evaluating: ",stock_trading_environment.total_account_value)
```

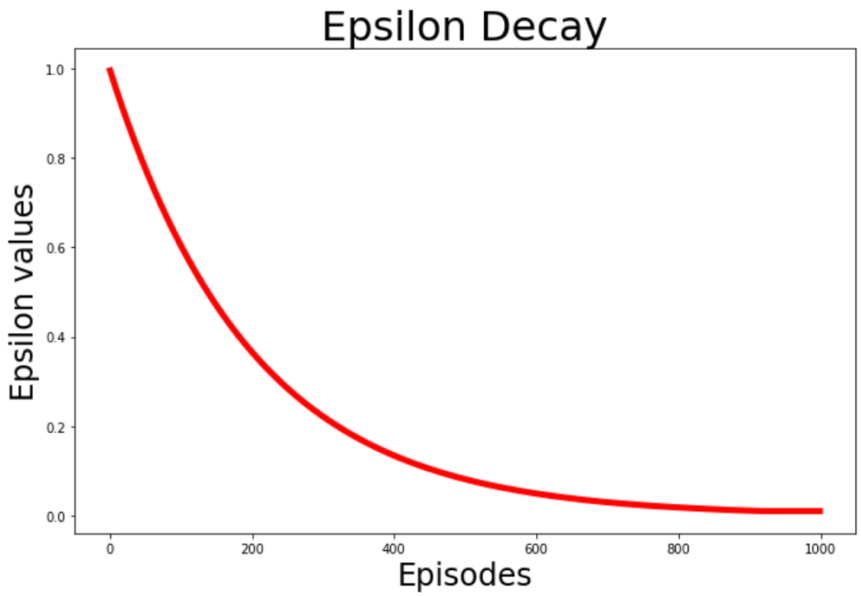



```
Total Account value after Evaluating:  173752.21279800002
```

Total Rewards per episode



Epsilon Decay per episode



7.References

1. Reinforcement Learning Tutorial | Reinforcement Learning Example Using Python | Edureka (https://www.youtube.com/watch?v=LzaWrmKL1Z4&t=1657s)

2. https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html

3. https://www.baeldung.com/cs/epsilon-greedy-q-learning