Part 1: Implement K-Means Clustering

Index

List of Figures

1] Overview:

The goal of the assignment is to learn the trends in stock price and perform a series of trades over a period of time and end with a profit. In each trade, you can either buy/sell/hold. You will start with an investment capital of $100,000 and your performance is measured as a percentage of the return on investment.

Q-Learning algorithm has to be used for reinforcement learning to train an agent to learn the trends in stock price and perform a series of trades. We have to implement Q-learning algorithm from scratch.

2] Dataset Description & Analysis:

The dataset provided is of the historical stock price for Nvidia for the last 5 years. The dataset has 1258 entries starting 10/27/2016 to 10/26/2021. The features include:

- Stock open price
- Intraday high and low
- Stock close price
- Adjusted closing price and
- volume of shares traded for the day

1 to 10 of 1258 entries Filter

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

Show 10 per page

1 2 10 100 120 126

Fig 1: Dataset

3] Environment Description:

The environment provided uses GYM which is an open-source Python library for developing and comparing reinforcement learning algorithms.
Environment mainly has 3 functions

- **reset() Function:**
    - This function resets the environment, that is it resets all the parameters
    - If the environment is in training mode it will calculate the price stock price changes, that is if the price has increased, decreased, or remained the same for maximum timesteps.

- And according to this change in stock price, it will create an observation vector that defines four states out of which one will be selected by the agent as the next state.
- The four sates are as follows:
  - 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.
- If the environment is in testing mode, the same steps are followed as training but instead of training data testing data is used to determine the observation vector based on price change
- The function returns observation

- **step() Function:**
  - This is the main function that implements based on the action taken by the agent, that is to buy, sell or hold the stock.
  - The action to be taken by the agent depends on the state provided by the observation vector.
    - If the action is to buy shares
      - If the agent already holds shares it is charged a penalty.
      - Based on the account value, number of shares the agent can buy is calculated.
      - The shares are bought and accordingly, all the parameters are updated and the agent is also rewarded.
    - If the action is to sell the shares
      - The shares are sold and accordingly the parameters are updated
      - If the agent makes a loss few reward points are deducted
    - If the action is to hold the shares
      - Similarly the parameters are updated and the reward is calculated.
  - If the environment is in testing mode, the same steps are followed as training but instead of training data testing data is used.
  - After the action has been taken the observation vector is calculated based on the stock price change and if the account holds stocks or not.
  - Total value in the account after the profit and loss based on the action taken is also calculated.

- The function returns the observation, reward, and boolean “true” if an episode is over else “false”.

- **plot () Function:**
  - This function basically plots the increase and decrease in the agent's total account value over time.

4] Implement the Q-learning algorithm:

- As per my research, Q-learning is a model-free reinforcement learning algorithm. It is a values-based learning algorithm, that is it updates the value function based on an equation, particularly Bellman equation.
- **Steps followed for Q-learning:**

1. **Initialize Q-table and other Parameters:**
   - Q-table has n columns, where n= number of actions. And m rows, where m= number of states
   - Epsilon[exploration probability] is declared as 1
   - Epsilon decay is set to 0.995, and the minimum Epsilon is set to 0.01
   - Gama has a value of 0.95
   - The agent is set to train for 500 episodes with a maximum of 200 iterations per episode incase the episode does not end.

```
self.env = environment

n_observations = self.env.observation_space.n
n_actions = self.env.action_space.n

self.Q_table = np.zeros((n_observations,n_actions)) #Initialize the Q-table to

self.epsilon = 1            #initialize the exploration probability to 1
self.epsilon_decay = 0.995 #exploartion decreasing decay for exponential
decreasing
 self.epsilon_min = 0.01    # minimum of exploration proba
 self.gamma = 0.95          #discounted factor
 self.lr = 0.1              #learning rate

 self.n_episodes = 5000           #number of episode we will run
 self.max_iter_per_episode = 200 #maximum of iteration per episode

 self.rewards_per_episode = []
 self.epsilon_per_episode = []
 self.episodes=[]
```

Fig 2: Initializing Q-table and other parameters

Train Function

2. **Chose & Perform Action:** This step continues for an undefined time and stops when all episodes are over
    - An action (a) in the state (s) is chosen based on the Q-Table.
    - The action is chosen based on a greedy strategy. That is in the beginning, the epsilon rates will be higher. The agent will explore the environment and randomly choose actions. Since the agent does not know anything about the environment. As the agent explores the environment, the epsilon rate decreases and the agent starts to exploit the environment.
    - During the process of exploration, the agent becomes more confident in estimating the Q-values.

```
if np.random.rand() <= self.epsilon:
     action = self.env.action_space.sample()
else:
     action = np.argmax(self.Q_table[current_state,:])
```

Fig 3: Chosing Action

3. **Update Q-table & Measure Reward:** The action chosen is fed to the environment which returns the observation and rewards.
    - The Q-table for this action and state is updated using the Bellman equation.
    - The reward received is added to calculate the cumulative reward per episode.
    - Next state received is set as the current state.
    - Epsilon is decayed at the end of the episode.

```
next_state, reward, done, info = self.env.step(action) #env runs
the chosen action

# Bellman equation
self.Q_table[current_state, action] = (1-self.lr) *
self.Q_table[current_state, action] +self.lr*(reward +
self.gamma*max(self.Q_table[next_state,:]))

total_episode_reward = total_episode_reward + reward

if done:
     break    #Break if episode finished
current_state = next_state
```

```
if self.epsilon > self.epsilon_min:
     self.epsilon *= self.epsilon_decay
```

Fig 4: Update Q-table & Measure Reward

Evaluate Function:

- Environment is first set to testing mode
- Best action from the Q-tabel is selected from the current state
- Action is fed to the environment which returns the next state and reward
- Finally the render function is called to plot total account value over time
- Evaluate function basically evaluates the trained agent for the best selected actions.

```
self.env.train = False
current_state = self.env.reset()
done = False

while not done:
    action = np.argmax(self.Q_table[current_state,:])
    next_state, reward, done, info = self.env.step(action)
            current_state = next_state
self.env.render()
```

Fig 5: Evaluation

5] Results and Plots:

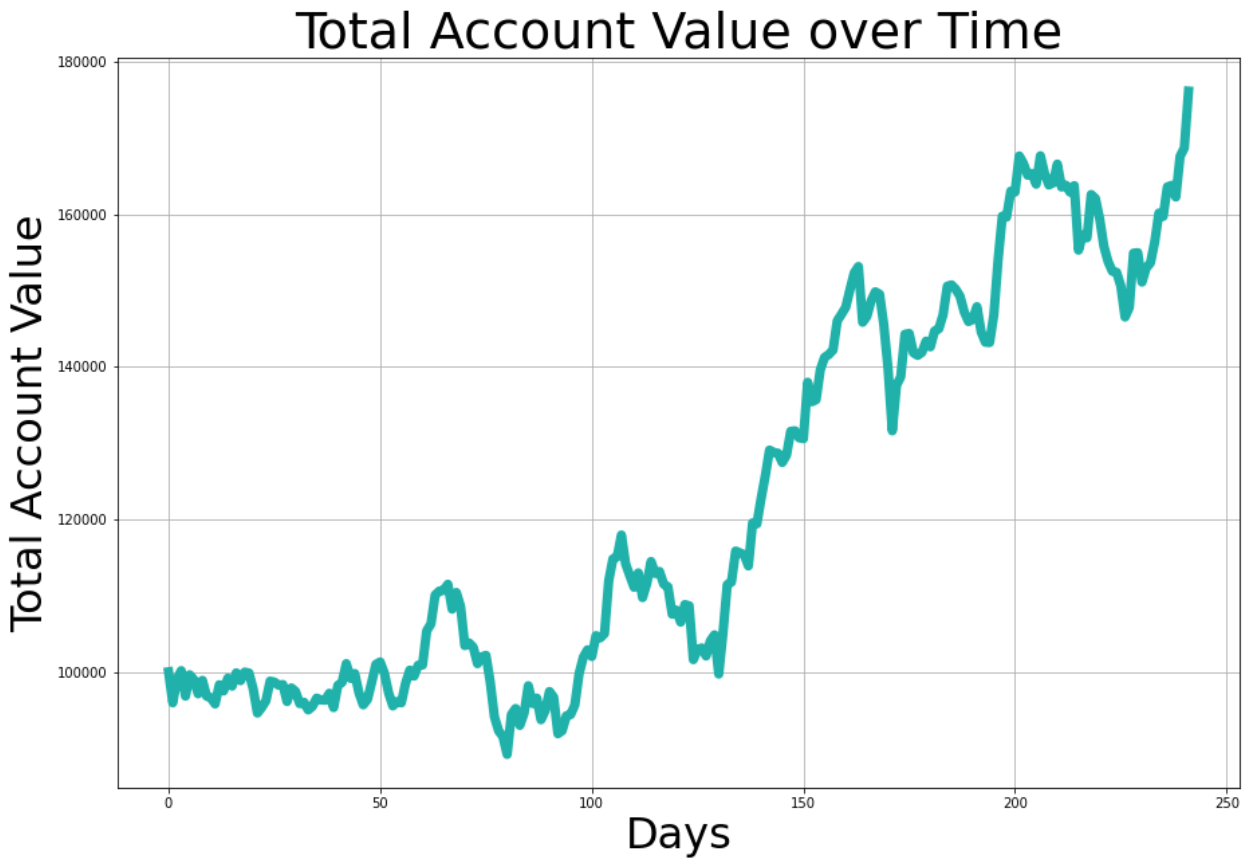

Fig 6: Graph for total account value over time

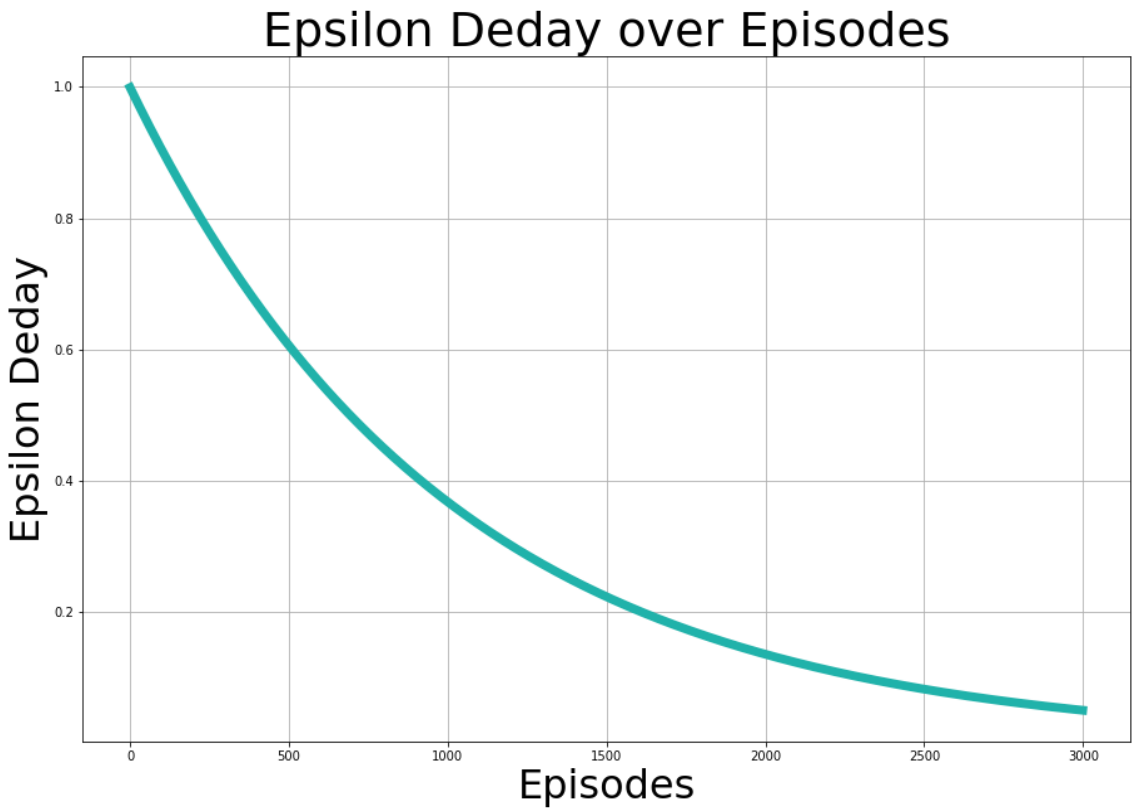

Fig 7: Graph for Epsilon decay over episodes

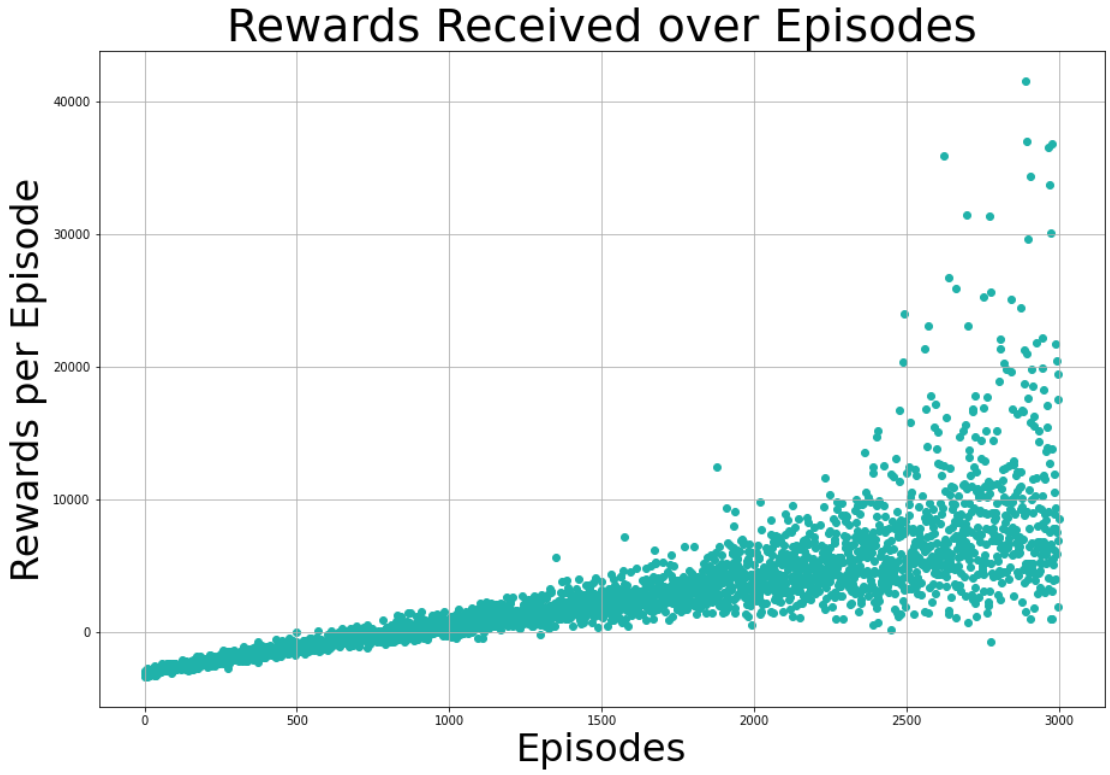

Fig 8: Graph for Total Reards over Episodes