ASSIGNMENT 1

DCSI 6650

Abstract

In this assignment the main goal is to do experiment with k armed bandit learning algorithms by implementing them from scratch and compare their performance

Group Members	Index No
Amarappulige Charuka Kavinda Priyankara	202296267
Chinonso Kingsley Imochukwu	202381038

Table of Contents

Part 1

Libraries used

In this exercise I have used two python libraries coding purposes.

- Numpy
- Matplotlib

Parameters

There are three main parameters used for governing the entire excelsis and rest of the parameters are function driven

- Number of arms (k)
- Steps of an episodes (steps)
- Number of episodes considered to get average performance (episodes)

Functions

Eight main functions are used to generate required output of the part 1 section and brief description and the algorithm used is explained in below

random_means

This function initialize the problem with generating random array of k elements with each element having a N(0, 1).

reward_func

This function return the reward value once it was given any value with normal distribution according to N(given value, 1)

optimal_action

This function evaluate the any action taken from an algorithm is optimal action or not according to the initially predicted k values for that specific episode.

greedy

Purpose of this function is to return the all the rewards and how many optimal action taken in the entire episode. The optimal action selection criteria are defined by the reinforcement learning algorithm and action selection is defined by;

$$NewEstimate = OldEstimate + StepSize(Target - OldEstimate)$$
$$Action = argmax(Estimate)\ or\ fist\ value\ if\ estimates\ are\ equal$$

epsilon_greedy

Here the function work similar to **greedy** function except the action is chosen according below logic

$$Action = \begin{cases} argmax(\ Estimate) & with\ prbability\ 1 - epsilon \\ a\ random\ action & with\ probability\ epsilon \end{cases}$$

greedy_with_SV

This function also works similar to **greedy** function except for we introduce optimistic initial value for all the estimates at starting point with experience we have with average reward gain.

gradient_bandit

In this function action is taken from numerical preference (H) instead of estimate and action is taken according to probability of (Pi) according to calculation given bellow. On each step, after selecting action At and receiving the reward Rt, the action preferences are updated

$$Pi_t = \frac{e^{H_t}}{\sum e^{H_t}}$$
$$H_{t+1}(At) = \ H_t(At) + \alpha(R_t - avg(R_t))$$
$$H_{t+1} = \ H_t - \alpha(R_t - avg(R_t))\ Pi_t$$

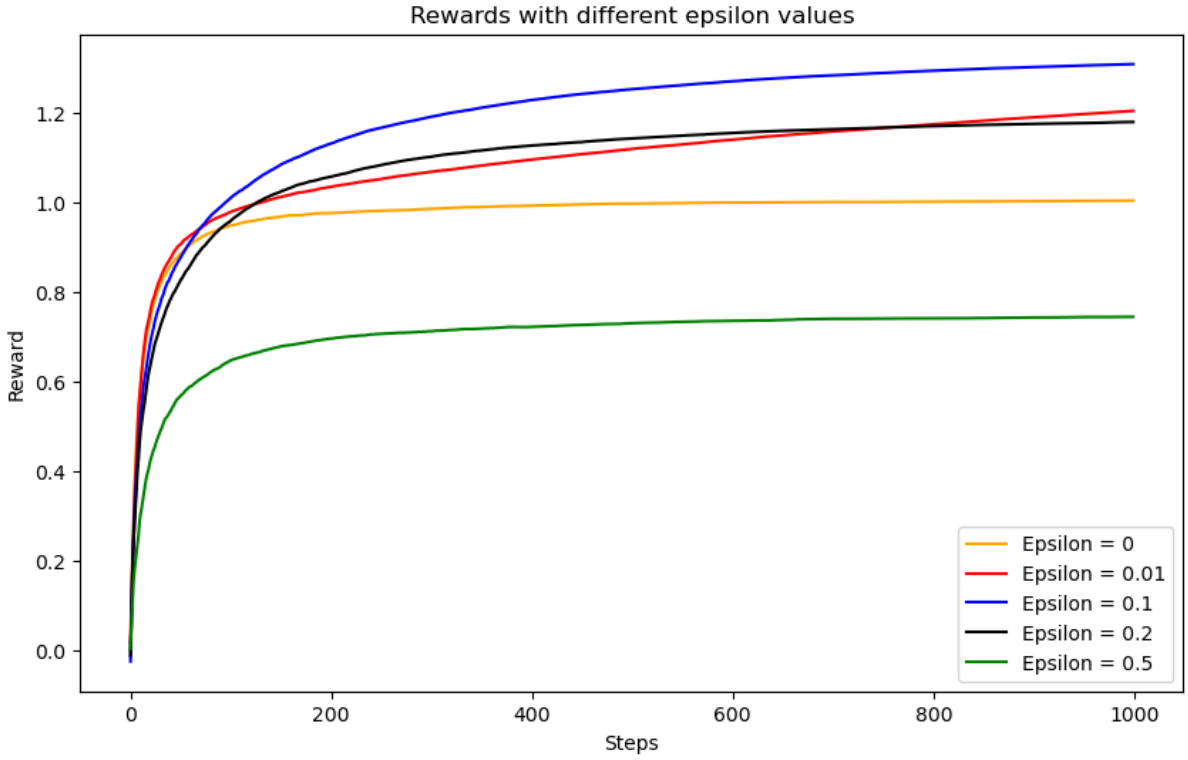
running_average

This function is used to calculate running average of steps with the all the reward taken from one episode.

Optimal epsilon

The value of epsilon for the second algorithm (epsilon_greedy function) has to select optimally in order to get highest reward and how sooner we want to get highest reward. Choosing higher epsilon value leads to argent to roam around with exploration rather increasing his reward. Lower epsilon value will lead argent to stick to local minimal for longer time than expected while increasing steps required to gain highest rewards.

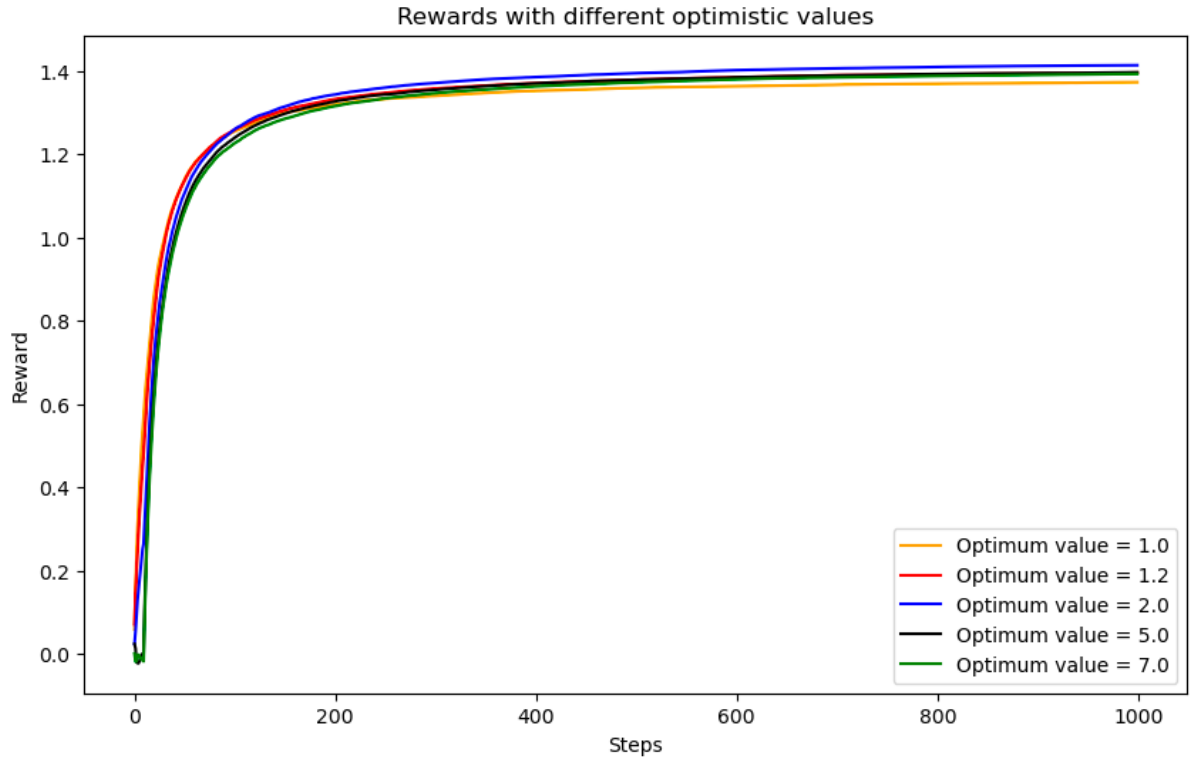
Running the second algorithm for multiple epsilon value will give a good understanding for its performance. In this experiment we have used epsilon values ranging from (0, 0.01, 0.1, 0.2, 0.5) for the performance analysis of 1000 steps with 1000 different episodes. Finally average reward gain is analyzed.



According to analysis it was visible to us that having higher epsilon value such as 0.5 gives the lowest gain. However, taking epsilon as 0 is not giving the maximum results either. Within the three values of epsilon 0.01, 0.1 and 0.2 it was obvious that 0.1 gives the highest reward. Due to this reason, We will use epsilon value as 0.1 for future calculations.

Optimistic initial value

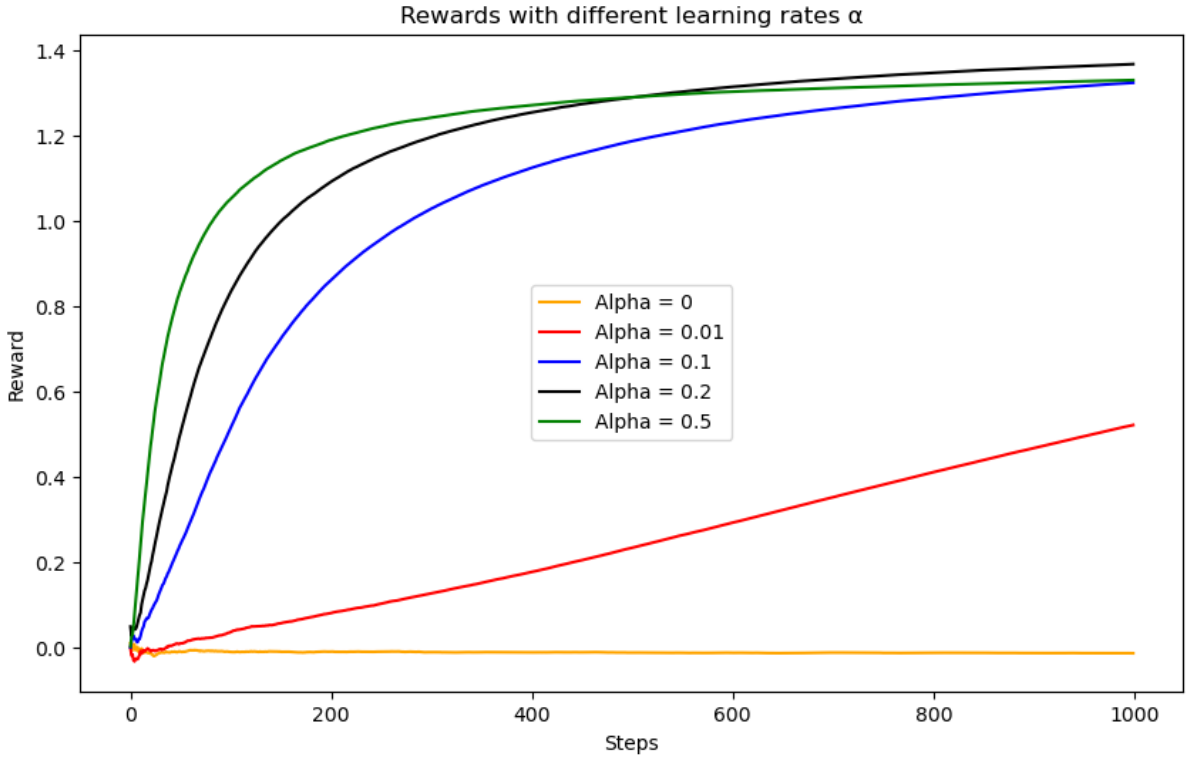
From the first analyses it was observed that reward value is hover around slightly above 1.2 region in all the algorithms. However, selection of accurate optimistic value will give a greater chance of acquiring the highest reward. Due to this reason second analysis were carried out to investigate the highest performing value for the optimistic greedy algorithm.



Here we have performed the 1000 steps for 1000 different episodes and average out results. From the analysis it came to realization that optimum value 2.0 gives the highest reward so far. I will be taking 2.0 value for each of the future simulations here onwards.

Optimal Learning Rate (α)

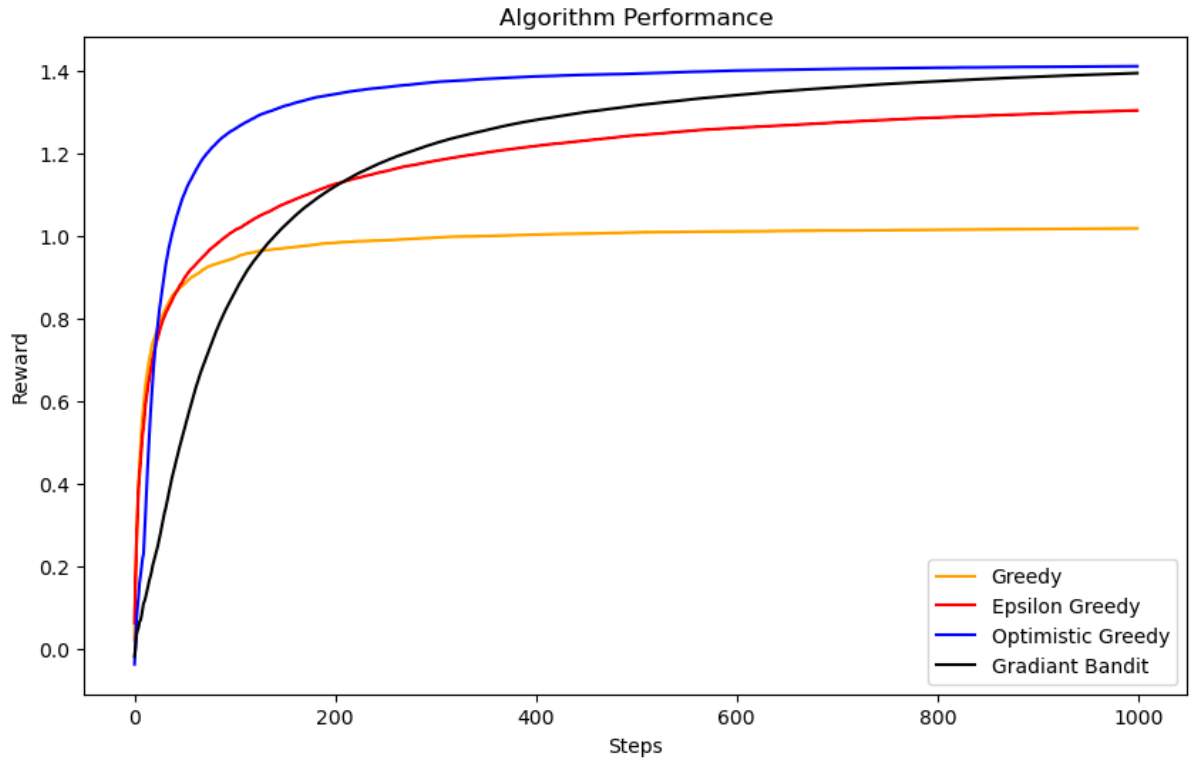
Selection of optimal learning rate for the gradient bandit algorithm is another opportunity to for reward maximization. In this segment we have used similar approach as epsilon where different alpha values (0, 0.01, 0.1, 0.2, 0.5) chosen for 1000 step episode taken average of 1000 episodes for evaluation.



According to results generated in above plot it can see that lower alpha values perform poorly while higher alpha values perform well. The highest alpha used (0.5) managed to get to highest reward in lesser time but saturated in lower value afterwards. The 0.1 alpha value give a promising trend in last episodes it took longer time to reach close to 0.5. Compared to them 0.2 alpha value gives adequate time to rise and give promising results than 0.1 and 0.5 at the end. Due to this reason for future analysis 0.2 value will be taken as optimum set point.

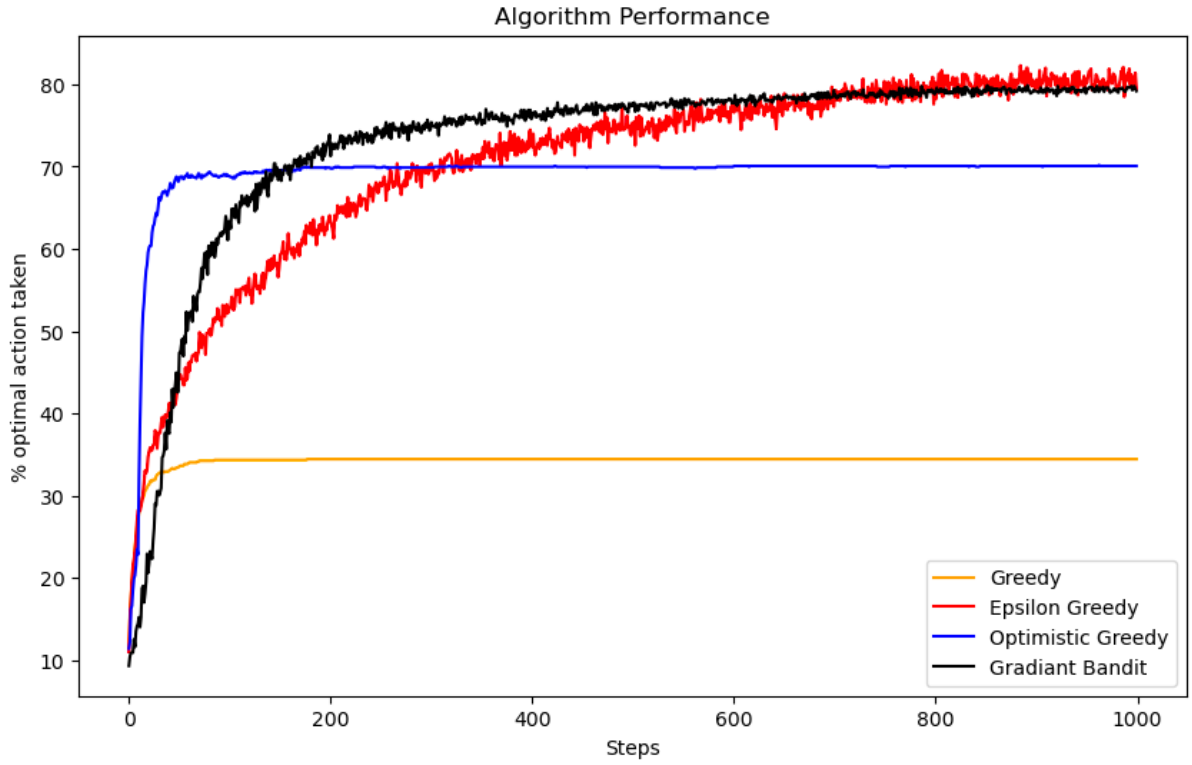
Algorithm performance

The performance of four algorithms according to selected parameters from above analysis is evaluated in 1000 bandit problems. Then the average reward acquired by each algorithm and percentage of time the optimal action taken have taken as the performance measuring criteria.



From the figure above it is visible that our greedy algorithm manages to rise faster compared to gradient bandits algorithm and settle in the reward region close to 1. However, the epsilon greedy algorithm managed to rise as fast as greedy algorithm but compared to greedy algorithm it continued to rise at steady rate till the end of the 1000 steps. The optimistic algorithm managed to surpass all the algorithms and rose to 1.4 reward region at very initial steps and saturated over there till the end of 1000 steps. Finally, the gradient bandit algorithm initially shown a lag to rise to higher rewards but surpass greedy and epsilon greedy algorithms' rewards around 150 and 200 steps respectively. Finally at the end it shows closing the gap towards optimistic greedy algorithm.

From the reward analysis it was visible that epsilon greedy, optimistic greedy and gradient bandit algorithm have a close performance. Due to this reason second set of data is required for performance evaluation. The percentage of time the optimal action taken is another good source for such requirement.



The figure above illustrates the percentage of time the optimal action taken by each step in 1000 episodes for each algorithm. It is obvious that greedy algorithm shows lowest percentage of optimal action with slightly above 35% of the time. Although, the optimistic greedy algorithm showed promising results with rewards, it managed to achieve correct action around 70% of the time. The epsilon greedy algorithm took longer time to take optimal action however at the final steps of the episode it managed to achieve correct action 80% of the time. Finally, the gradient bandit algorithm managed pick optimum action 70% of the time with less than 200 steps and continued to rise till the last steps.

From the above analysis I can conclude that the person who want quick reward and willing to compromise accuracy can go with optimistic greedy algorithm. While who is interested in higher accuracy of action selection and delayed returns can choose either epsilon greedy or gradient bandit algorithms.

Running the steps up to 10,000 will give good estimation for the people who are interested in long-term return. The code for the analysis saved in the below path named as 'Assignment1.1.ipynb'. Running the entire python notebook can recreate the results. Changes can be done to parameters for further analysis.

Link- https://github.com/CharukaP/Reinforcement_Learning.git

Part 2

In real world fixed values for rewards are difficult to observe there is always possibility to change in values with time. In this section we will explore such changes and how the algorithm manages to perform with those changes.

Change in Parameters

We are exploring three different changes to our mean parameters in order to mimic real world scenarios and introduce those functions to our original code.

- Drift change
- Reverting change
- Abrupt change

First change we can introduce is drift change here we consider drift in mean values with each step of the actions. Change in mean value is defined as below formula and according to that drift_change function is added to the code.

$$\mu_t = \mu_{t-1} + \epsilon_t \mid \epsilon_t \in N(0, 0.001^2)$$

Second changing behavior is reverting change here we consider slightly bigger change in mean values than first instance. Change in mean is explained well in below formula. Similarly reverting_change function is introduced for the code to accommodate change

$$\mu_t = K\,\mu_{t-1} + \epsilon_t \mid \epsilon_t \in N(0, 0.001^2), K = 0.5$$

The last change was abrupt change. Here we permute the element in mean values of k-arms with probability of 0.005. Although its small number change in value have a huge impact on reward collection.

With all these changes new parameter introduced to all the four algorithms with mean changing parameter where input of 0 to 3.

- 0 = No mean change
- 1 = Drift change
- 2 = Reverting change
- 3 = Abrupt change

Performance evaluation of non-stationary adaption

Adaptation to non-stationary problem of different algorithm is a paramount requirement in this study. Such comparison requires different approach due non-stationary behavior of reward distribution. Hence, we compare algorithm performance with the distribution of last reward of each episode keeping the step size of each episode to 10,000 iterations. Similarly, the total episodes count for this comparison kept as 1000 as previously. The main three algorithms evaluated in study are.

1. Optimistic greedy (decreasing step size)
2. ϵ - greedy with fixed step size (step size = 0.1)
3. ϵ - greedy with decreasing step size

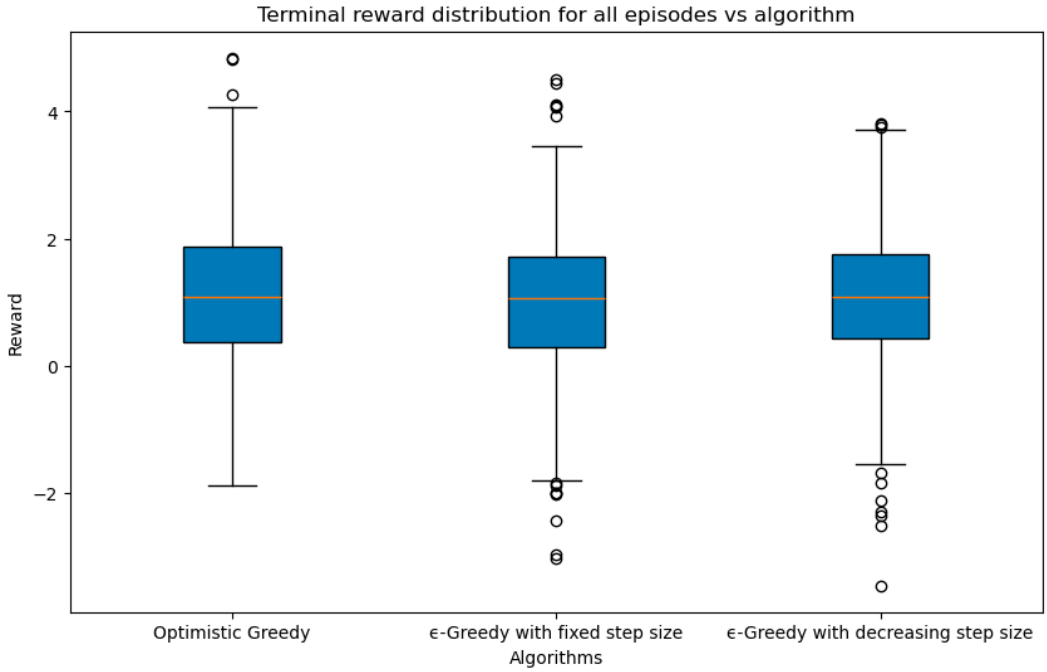
Furthermore, two different initiation steps carried out in order to analyze results. This will let the reader a better understanding of the performance of algorithm.

1. Same starting mean for all the episodes (Randomly generated mean with seed=100)
2. Same mean for each bandit different value for each episode (randomly generated μ_i where $\mu_1=\mu_2=\mu_3=\mu_4.....=\mu_{10}$)

1.Same starting mean performance

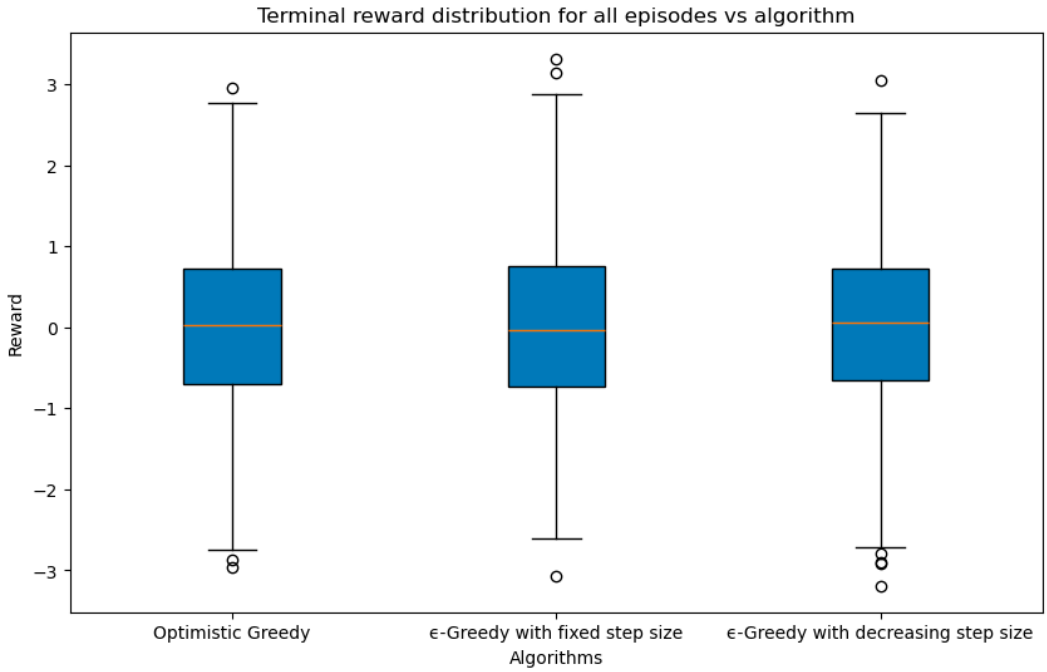
Here illustrates the performance for each algorithm in three different changes in parameters as described in parameter section.

Drift change



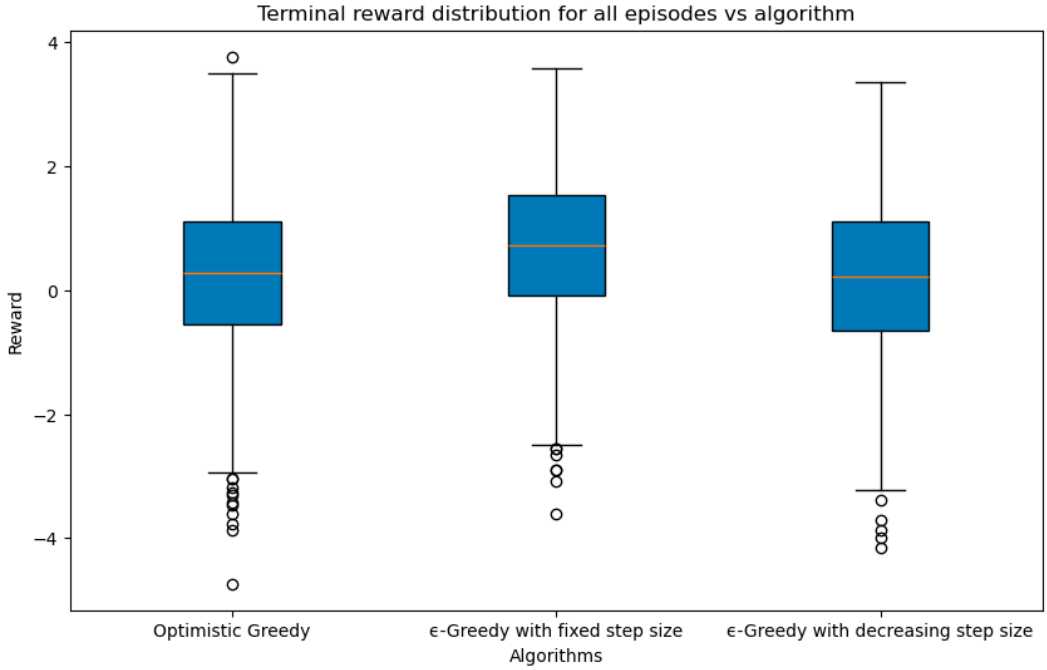
Mean values - [1.12886891, 1.00833572, 1.06364015]

Reverting change



Mean values - [0.0190896, 0.01604607, 0.02660606]

Abrupt change



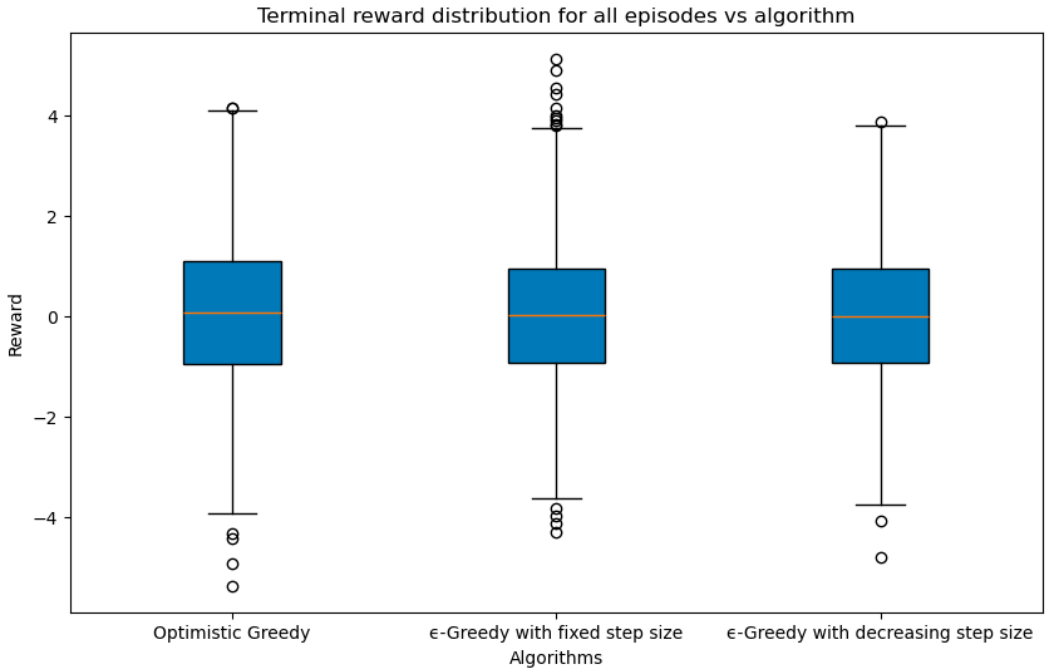
Mean values - [0.22323222, 0.69393108, 0.20836628]

In conclusion it can be observed that all three algorithms perform slightly poorer than without any mean changes as in part 1. However, when there is a drift change it can observe that optimistic greedy algorithm outperforms the other two. Reverting change is the algorithm that shown the poorest results with slight changes between algorithms. According to result the ϵ-greedy with decreasing step size algorithm is the one shown the maximum gain. Finally, when there is abrupt change in the system ϵ-greedy with fixed step size algorithm outperform the rest

2.Same mean for each bandit performance

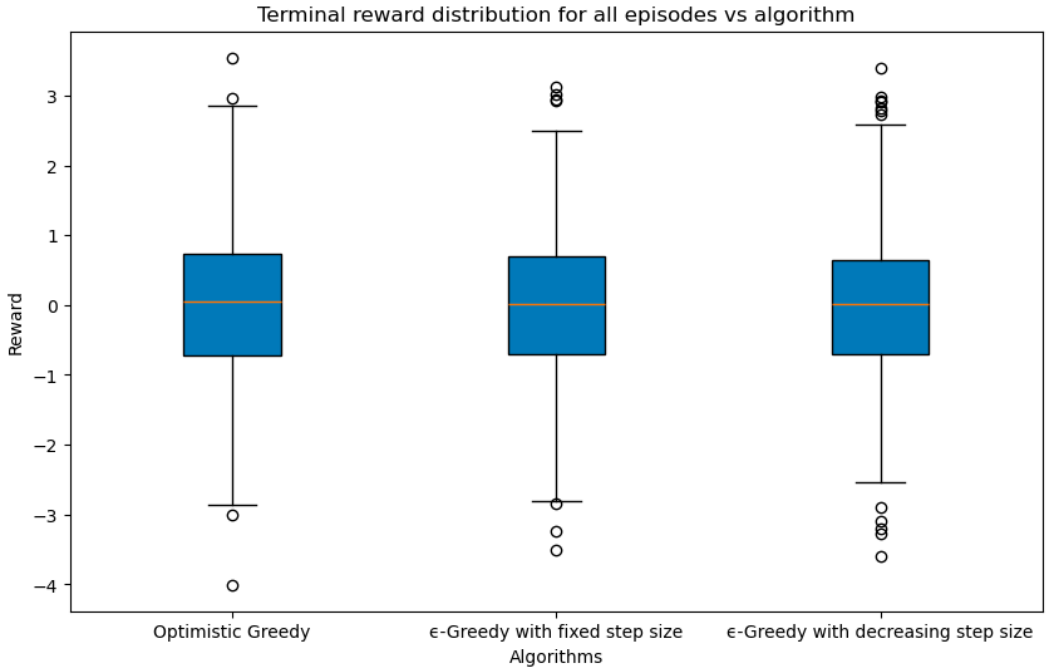
Here illustrates the performance for each algorithm in three different changes in parameters similar to above section for second mean generating methodology.

Drift change



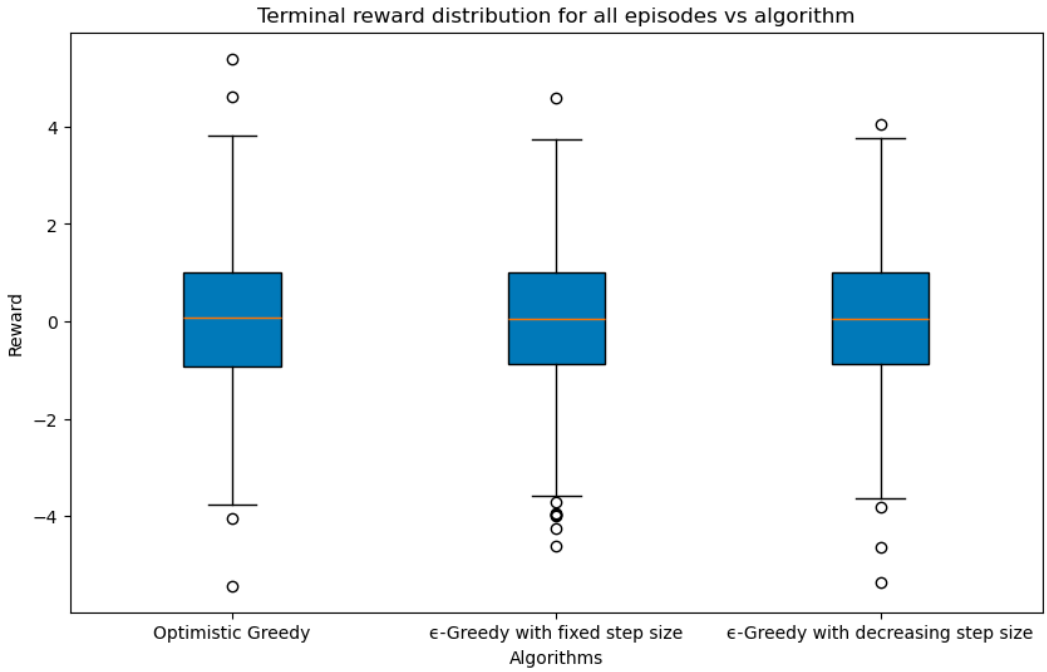
Mean values - [0.05995532, 0.02791486, -0.01034939]

Reverting change



Mean values - [0.01799785, -0.0250348, -0.01218333]

Abrupt change



Mean values - [0.05885393, 0.01777314, 0.05182503]

In this methodology the performance of algorithm has degraded heavily. The average reward distribution of all three instances hover around ±0.06 region, making it less efficient for this kind of operations. However, out of all the three changes in mean values, optimistic greedy algorithm managed to give the best results.

Further analysis can be carried out for these algorithms to analyze performance and find out suitability. The code is freely available in below link under the name ‘Assignment1.2.ipynb’. Running entire notebook with different parameters mentioned above chapters will recreate all the results we have discussed so far.

Link- https://github.com/CharukaP/Reinforcement_Learning.git