Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Reinforcement Learning

AY: 2022 - 23

Experiment 1

Exploration Exploitation Dilemma

AIM :

a) To solve the exploration exploitation dilemma using epsilon greedy strategy
b) To understand the effect of epsilon by comparing the different values of epsilon

THEORY:

With partial knowledge about future states and future rewards, our reinforcement learning agent will be in a dilemma on whether to exploit the partial knowledge to receive some rewards or it should explore unknown actions which could result in much larger rewards.

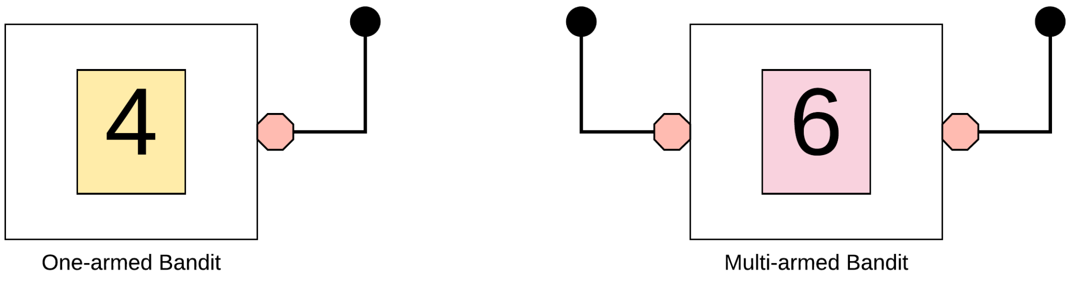
- Exploitation: Make the best decision given current information. The best long-term strategy may involve short-term sacrifices
- Exploration Gather more information. Gather enough information to make the best overall decisions

However, we cannot choose to explore and exploit simultaneously. In order to overcome the Exploration-Exploitation Dilemma, we use the Epsilon Greedy Policy.

MULTI ARMED BANDIT PROBLEM (MAB)

In a multi-armed bandit problem (MAB) (or n-armed bandits), an Agent makes a choice from a set of actions. This choice results in a numeric reward from the Environment based on the selected action. In this specific case, the nature of the Environment is a stationary probability distribution. By stationary, we mean that the probability distribution is constant (or independent) across all states of the Environment. In other words, the probability distribution is unchanged as the state of the Environment changes. The goal of the Agent in a MAB problem is to maximize the rewards received from the Environment over a specified period.

The MAB problem is an extension of the "one-armed bandit" problem, which is represented as a slot machine in a casino. In the MAB setting, instead of a slot machine with one-lever, we have multi-levers. Each lever corresponds to an action the Agent can play. The goal of the Agent is to make plays that maximize its winnings (i.e. rewards) from the machine. The Agent will have to figure out the best levers (exploration) and then concentrate on the levers (exploitation) that will maximize its returns (i.e. the sum of the rewards).



Left: One-armed bandit. The slot machine has one lever that returns a numerical reward when played.

Right: Multi-armed bandits. The slot machine has multiple (n) arms, each returning a numerical reward when played. In a MAB problem, the reinforcement agent must balance exploration and exploitation to maximize returns.

Epsilon-greedy

The agent does random exploration occasionally with probability ἐ and takes the optimal action most of the time with probability 1− ἐ.

Epsilon greedy method. At each step, a random number is generated by the model. If the number was lower than epsilon in that step (exploration area) the model chooses a random action and if it was higher than epsilon in that step (exploitation area) the model chooses an action based on what it learned.

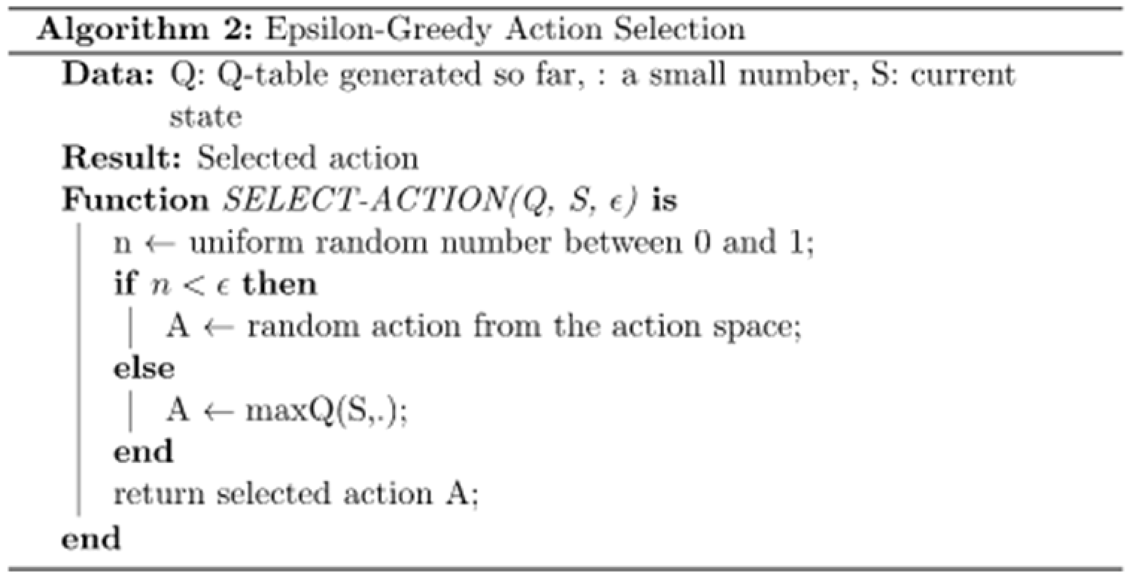
Usually, epsilon is set to be around 10%.

Epsilon-Greedy can be represented as follows:

$$A_t \leftarrow \begin{cases} argmax\ Q_t(a) & with\ probability\ 1-\epsilon \\ a \sim Uniform(\{a_1 \ldots a_k\}) & with\ probability\ \epsilon \end{cases}$$

The Action that the agent selects at time step t, will be a greedy action (exploit) with probability (1-epsilon) or may be a random action (explore) with probability of epsilon.

ALGORITHM:



```
Algorithm 2: Epsilon-Greedy Action Selection
  Data: Q: Q-table generated so far, : a small number, S: current
        state
  Result: Selected action
  Function SELECT-ACTION(Q, S, $\epsilon$) is
      n ← uniform random number between 0 and 1;
      if $n < \epsilon$ then
          A ← random action from the action space;
      else
          A ← maxQ(S,.);
      end
      return selected action A;
  end
```

LAB ASSIGNMENT TO DO:

1. Create a multi armed bandit agent which would estimate the win rate using the epsilon greedy strategy
2. Understand the effect of the value of epsilon on the win rate by comparing the win rates corresponding to different values of epsilon. Hence draw conclusions.

Sarvayga Singh

K/K-1

60009200030

ML-3 LAB-1

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:

```
def select_action(epsi,epochs,alpha,Q):
  action=[]
  for i in range (epochs):
    n=np.random.random()
    arm=np.random.randint(0,2)
    if( n<epsi):
      action.append(arm+1)
      if(arm==0):
        Q[0].append(Q[0][-1]+(alpha*(R[arm]-Q[0][-1])))
      else:
        Q[1].append(Q[1][-1]+(alpha*(R[arm]-Q[1][-1])))

    else:
      r1=max(Q[0])
      r2=max(Q[1])
      if(r1>r2):
        Q[0].append(Q[0][-1]+(alpha*(R[arm]-Q[0][-1])))
      else:
        Q[1].append(Q[1][-1]+(alpha*(R[arm]-Q[1][-1])))
      # print(action)
  return Q
```

In [3]:

```
R=[ np.random.randint(1,100), np.random.randint(1,100) ]
Q=[[np.random.random()],[np.random.random()]]
print("Reward ",R)
print("Estimated ",Q)
```

```
Reward  [52, 29]
Estimated  [[0.919106165354691], [0.623350124878374]]
```

In [4]:

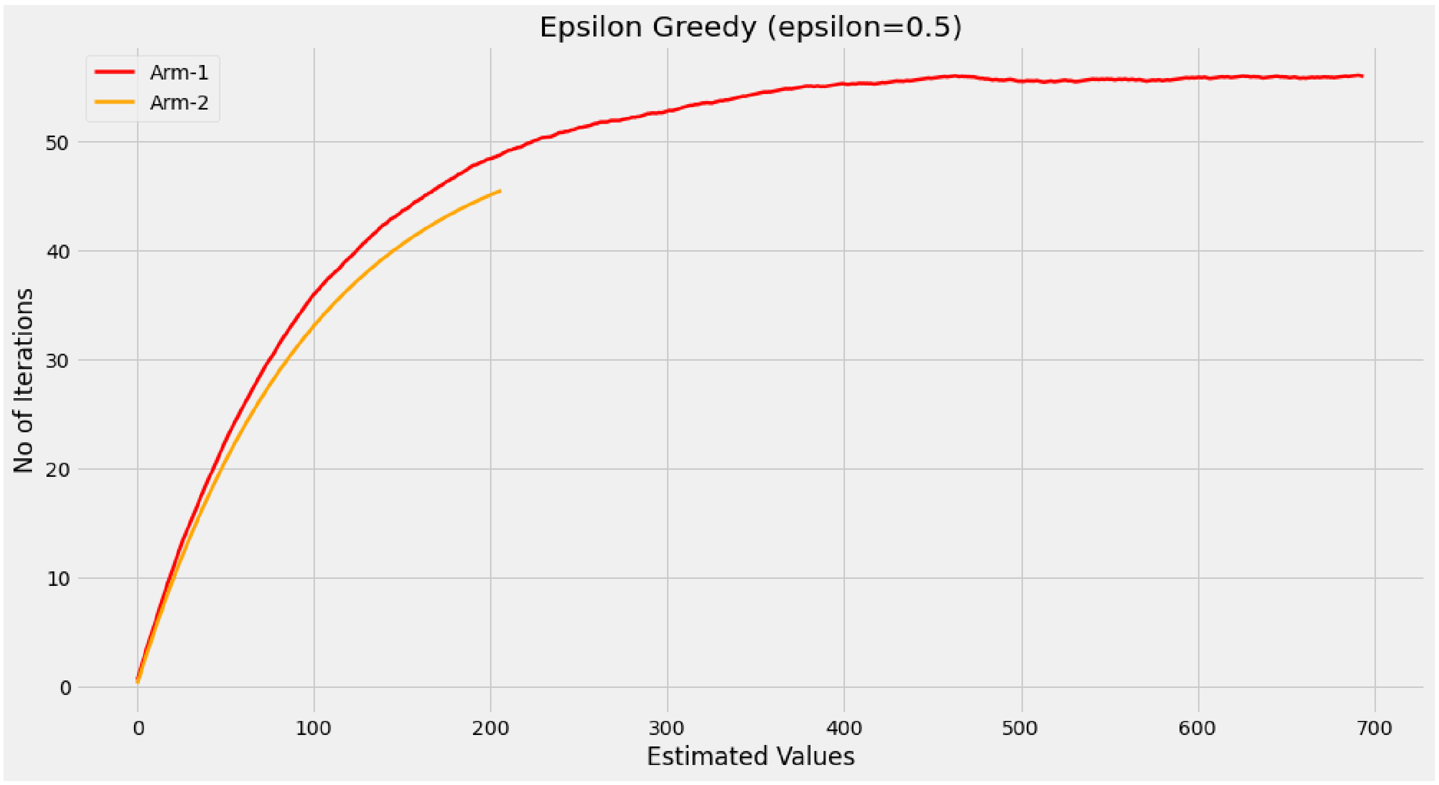
```
# %% capture
value1 = select_action(0.5,100,0.01,Q)
value1[0]
```

Out[4]:

```
[0.919106165354691,
 1.1999151037011442,
 1.7079159526641328,
 1.9808367931374915,
 2.4810284252061168,
 2.976218140954056,
 3.4664559595445152,
 3.95179139994907,
 4.202273485949579,
 4.6802507510900835,
 5.153448243579183,
 5.621913761143391,
 6.085694623531957,
 6.544837677296638,
 6.999389300523672,
 7.219395407518435,
 7.66720145344325,
 7.880529438908818,
 8.09172414451973,
 8.530806903074533,
 8.965498834043787,
 9.16584384570335,
 9.594185407246316,
 9.788243553173853,
 9.980361117642115,
 10.170557506465693,
 10.588851931401036,
 11.002963412087025,
 11.182933777966156,
 11.361104440186494,
 11.76749339578463,
 12.169818461826784,
 12.568120277208516,
 12.96243907443643,
 13.352814683692065,
 13.509286536855145,
 13.894193671486594,
 14.275251734771729,
 14.652499217424012,
 15.025974225249772,
 15.165714482997275,
 15.534057338167303,
 15.89871676478563,
 16.259729597137774,
 16.617132301166397,
 16.74096097815473,
 17.093551368373184,
 17.212615854689453,
 17.560489696142557,
 17.904884799181133,
 18.01583595118932,
 18.125677591677427,
 18.46442081576065,
 18.569776607603043,
 18.674078841527013,
 19.00733805311174,
 19.337264672580623,
 19.663892025854818,
 19.75725310559627,
 19.849680574540308,
 19.941183768794904,
 20.031771931106956,
 20.121454211795886,
 20.440239669677926,
 20.755837272981147,
 20.838278900251336,
 20.91989611124882,
 21.230697150136333,
 21.53839017863497,
 21.84300627684862,
 22.144576214080132,
 22.213130451939332,
 22.510999147419938,
 22.575889155945738]
```

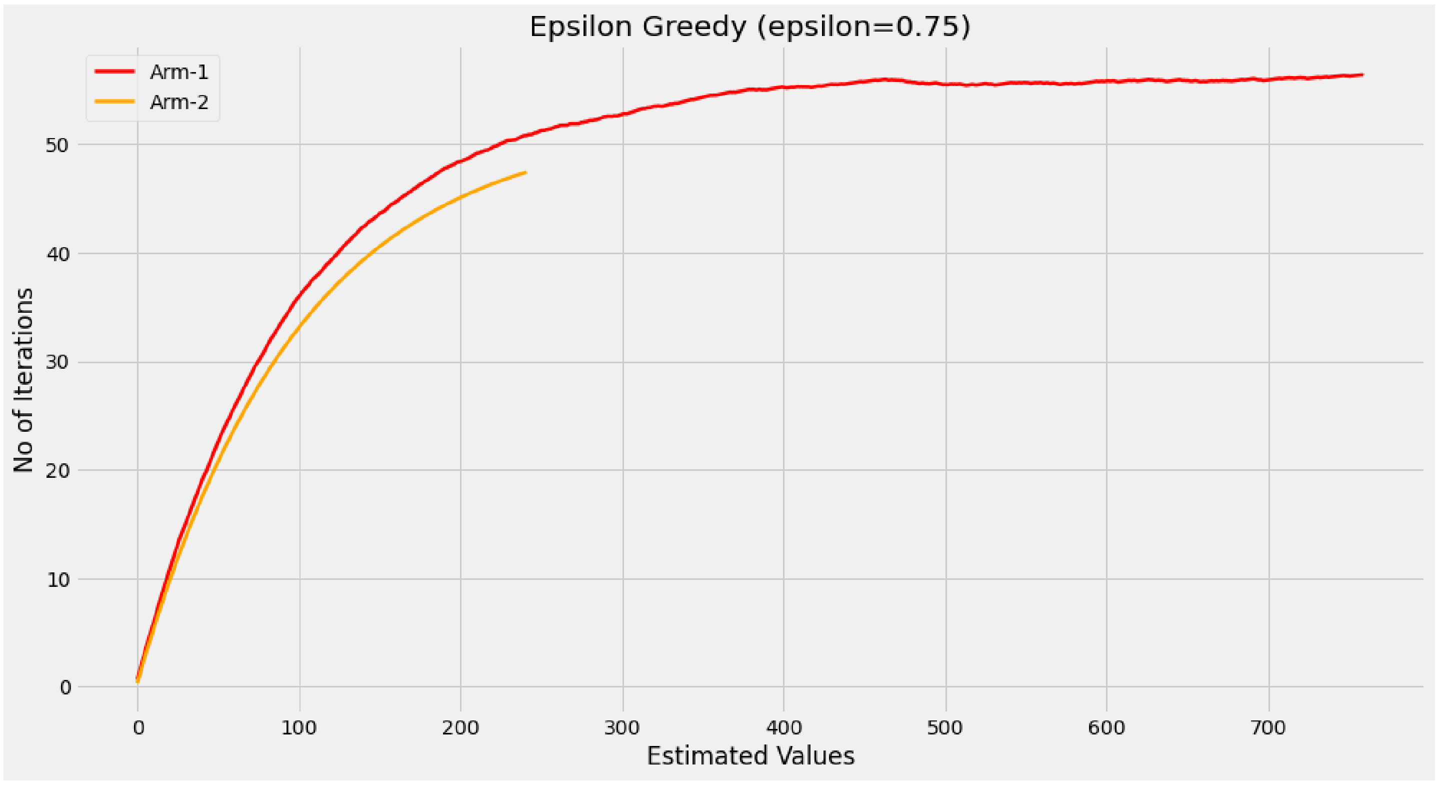
In [16]:

```
plt.figure(figsize=(15,8))
plt.plot(value1[0],label="Arm-1",linewidth=2.5,color="red")
plt.plot(value1[1],label="Arm-2",linewidth=2.5,color="orange")
plt.xlabel("Estimated Values")
plt.ylabel("No of Iterations")
plt.style.use('fivethirtyeight')
plt.title("Epsilon Greedy (epsilon=0.5)")
plt.legend()
plt.show()
```



In [17]:

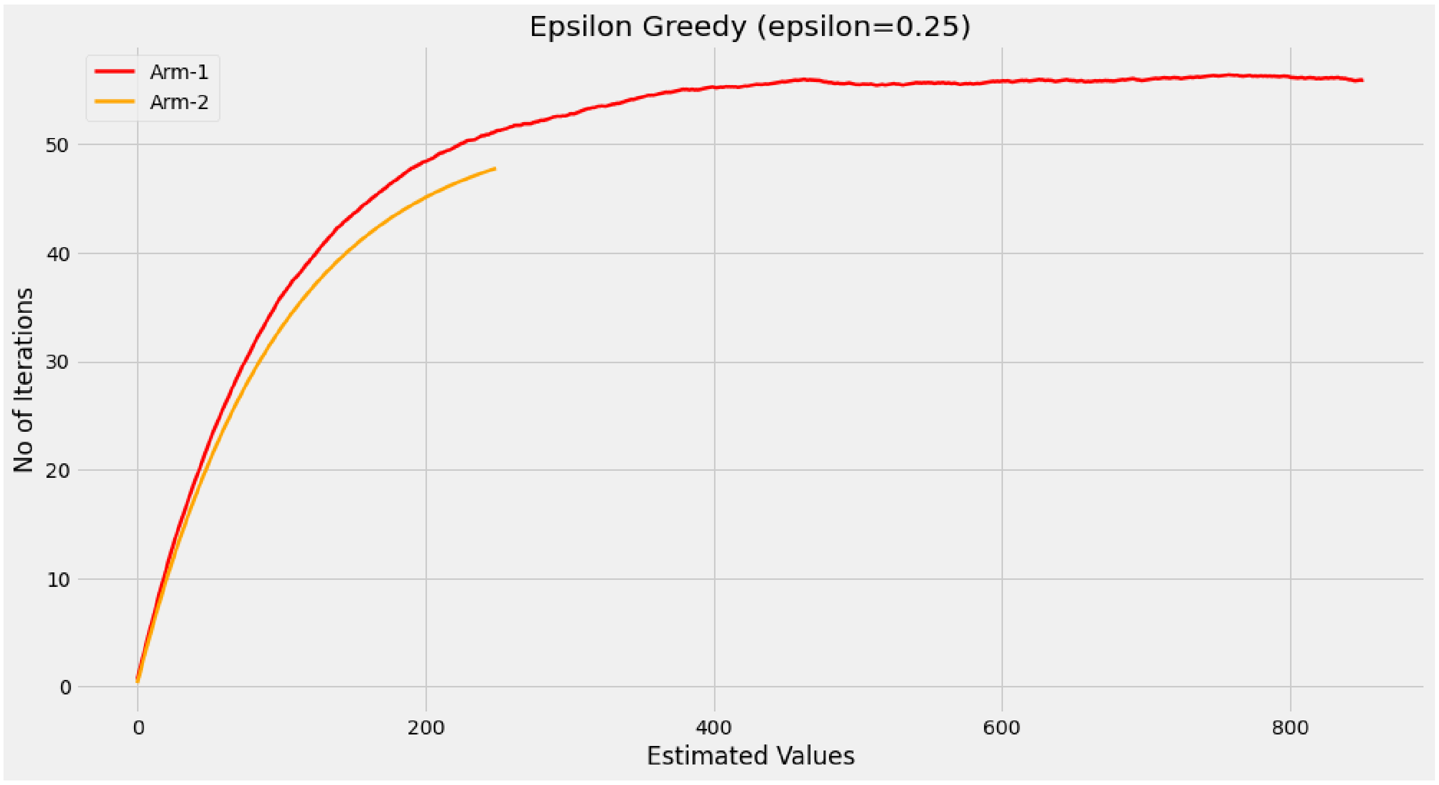
```
value2 = select_action(0.75,100,0.01,Q)
plt.figure(figsize=(15,8))
plt.plot(value2[0],label="Arm-1",linewidth=2.5,color="red")
plt.plot(value2[1],label="Arm-2",linewidth=2.5,color="orange")
plt.xlabel("Estimated Values")
plt.ylabel("No of Iterations")
plt.style.use('fivethirtyeight')
plt.title("Epsilon Greedy (epsilon=0.75)")
plt.legend()
plt.show()
```



In [18]:

```
value3 = select_action(0.25,100,0.01,Q)
```

```
plt.figure(figsize=(15,8))
plt.plot(value3[0],label="Arm-1",linewidth=2.5,color="red")
plt.plot(value3[1],label="Arm-2",linewidth=2.5,color="orange")
plt.xlabel("Estimated Values")
plt.ylabel("No of Iterations")
plt.style.use('fivethirtyeight')
plt.title("Epsilon Greedy (epsilon=0.25)")
plt.legend()
plt.show()
```



In [5]:

```
!jupyter nbconvert python --to html ML_3_Lab_1.ipynb
```

```
usage: jupyter-nbconvert [-h] [--debug] [--show-config] [--show-config-json]
                         [--generate-config] [-y] [--execute] [--allow-errors]
                         [--stdin] [--stdout] [--inplace] [--clear-output]
                         [--no-prompt] [--no-input]
                         [--allow-chromium-download]
                         [--disable-chromium-sandbox] [--show-input]
                         [--embed-images] [--log-level NbConvertApp.log_level]
                         [--config NbConvertApp.config_file]
                         [--to NbConvertApp.export_format]
                         [--template TemplateExporter.template_name]
                         [--template-file TemplateExporter.template_file]
                         [--theme HTMLExporter.theme]
                         [--writer NbConvertApp.writer_class]
                         [--post NbConvertApp.postprocessor_class]
                         [--output NbConvertApp.output_base]
                         [--output-dir FilesWriter.build_directory]
                         [--reveal-prefix SlidesExporter.reveal_url_prefix]
                         [--nbformat NotebookExporter.nbformat_version]
                         [extra_args ...]
jupyter-nbconvert: error: unrecognized arguments: ML_3_Lab_1.ipynb
```

In [ ]: