**Experiment No.02**

PART A

(PART A: TO BE REFFERED BY STUDENTS)

**A.1 Aim: Implementation of Multi-Arm Bandit (MAB) Problem**

1. To implement MAB problem using pure exploitation algorithm.
2. To implement MAB problem using pure exploration algorithm
3. To implement MAB problem using Fixed Exploration followed by Exploitation
4. To implement MAB problem using Greedy algorithm
5. To apply Upper Confidence Bound(UCB) in the implementation done in part d
6. To analyse the algorithms and compare them in terms of rewards, regrets and complexity

**A.2 Prerequisite:**

Concept of Multi-Arm Bandit, Exploration, Exploitation, Greedy, UCB

**A.3 Learning Outcome:**

After completing thisexperiment you will be able to-

* Comprehend the fundamental concept of Multi-Arm Bandit Problem
* Implementation of various strategies
* Fine-tuning of to balance exploration and exploitation
* Comparative analysis of all the algorithms to identify algorithm with maximum award

**A.4 Theory:**

**A.4.1 Multi-Arm Bandit Problem**

The Multi-Arm Bandit (MAB) problem is a classic reinforcement-learning problem that explores the trade-off between exploitation (choosing the best-known option) and exploration (trying new options to discover their potential). In this problem, an agent is faced with multiple slot machines (referred to as arms), each with an unknown probability of winning. The agent's goal is to maximize the total reward over a series of trials by deciding which arm to pull in each trial.

**Exploration**

**Exploration** refers to the strategy where an agent tries out different actions to gather more information about their potential rewards. In the context of the Multi-Arm Bandit problem, this means pulling different arms to learn more about their reward distributions.

**Exploitation**

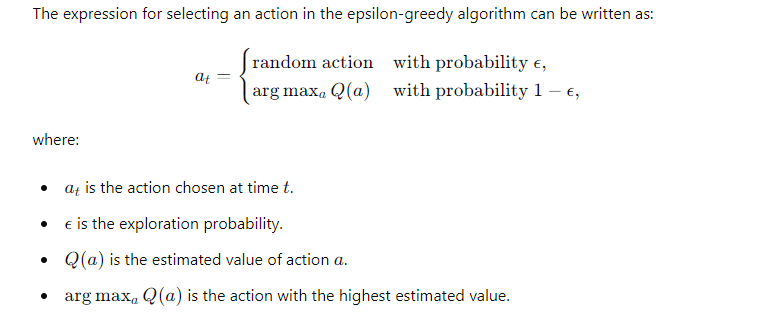
**Exploitation** involves selecting the action that is currently believed to be the best based on experiences. In the Multi-Arm Bandit problem, this means always pulling the arm with the highest estimated reward.

**Epsilon-Greedy Algorithm**

The **epsilon-greedy algorithm** is a simple and effective strategy that combines both exploration and exploitation. The key idea is to choose the best-known action most of the time, but occasionally explore other actions. This balance is controlled by a parameter, epsilon (ε)

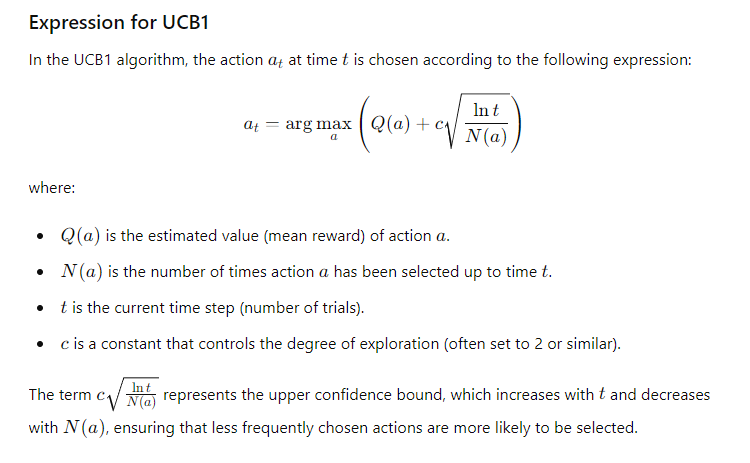
**How Epsilon-Greedy Works**

* **Parameter ε**: A value between 0 and 1 that determines the likelihood of exploration versus exploitation. For example, ε = 0.1 means that there is a 10% chance of exploring and a 90% chance of exploiting.
* **Decision Rule**:
  + With probability ϵ, choose a random action (exploration).
  + With probability 1−ϵ, choose the action with the highest estimated reward (exploitation).
* **Update**: After selecting an action and receiving a reward, update the estimated rewards based on the observed outcomes.



**UCB**

The UCB algorithm balances exploration and exploitation by considering the uncertainty in the estimates of the rewards. It selects the arm with the highest upper confidence bound.



**A.5 Task to be completed:**

1. Take value of arm as n=5 and number of trials t =500, Positive Reward =1, Negative Reward =0
   1. Implement Exploration
   2. Implement Exploitation
   3. Implement Fixed Exploration and then Exploitation
   4. Implement greedy
   5. Fine tune the value of and encapsulate your observations
   6. Compare the algorithms in terms or complexity, value and regret
   7. Incorporate UCB in greedy and state your opinion
2. Take different values of n and t and summarize your observations
3. Give two real world applications of MAB

**References**

[**https://www.analyticsvidhya.com/blog/2018/09/reinforcement-multi-armed-bandit-scratch-python/**](https://www.analyticsvidhya.com/blog/2018/09/reinforcement-multi-armed-bandit-scratch-python/)

[**https://www.geeksforgeeks.org/multi-armed-bandit-problem-in-reinforcement-learning/**](https://www.geeksforgeeks.org/multi-armed-bandit-problem-in-reinforcement-learning/)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**PART B**

(PART B: TO BE COMPLETED BY STUDENTS)

**(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Portal/MS Teams assignment link at the end of the practical)**

|  |  |
| --- | --- |
| Roll No. C052 | Name: Drumil Kotecha |
| Program: BTI | Division: B |
| Batch: B2 | Date of Experiment: 21/1/25 |
| Date of Submission: 15/1/25 | Grade : |

**B.1 Tasks given in PART A to be completed here**

*(****Students must write the answers of the task(s) given in the PART A /Students must copy the code, output screenshots here based on the task(s) given in section A.5)***

import numpy as np

import matplotlib.pyplot as plt

# Parameters

n = 5  # Number of arms

t = 500  # Number of trials

positive\_reward = 1

negative\_reward = 0

true\_rewards = np.random.rand(n)

# Function to simulate the environment

def simulate\_environment(arm):

    return np.random.choice([positive\_reward, negative\_reward], p=[true\_rewards[arm], 1 - true\_rewards[arm]])

# Exploration and Exploitation

def exploration\_exploitation():

    rewards = np.zeros(t)

    for trial in range(t):

        if trial < n:

            arm = trial

        else:

            arm = np.argmax(rewards[:trial])

        rewards[trial] = simulate\_environment(arm)

    return rewards

# Fixed Exploration then Exploitation

def fixed\_exploration\_exploitation(fixed\_trials=100):

    rewards = np.zeros(t)

    for trial in range(t):

        if trial < fixed\_trials:

            arm = trial % n

        else:

            arm = np.argmax(rewards[:fixed\_trials])

        rewards[trial] = simulate\_environment(arm)

    return rewards

# Epsilon-Greedy Algorithm

def epsilon\_greedy(epsilon=0.1):

    rewards = np.zeros(t)

    counts = np.zeros(n)

    estimates = np.zeros(n)

    for trial in range(t):

        if np.random.rand() < epsilon:

            arm = np.random.randint(n)  # Explore

        else:

            arm = np.argmax(estimates)  # Exploit

        reward = simulate\_environment(arm)

        counts[arm] += 1

        estimates[arm] += (reward - estimates[arm]) / counts[arm]

        rewards[trial] = reward

    return rewards

# Epsilon-Greedy with UCB

def epsilon\_greedy\_ucb(epsilon=0.1):

    rewards = np.zeros(t)

    counts = np.zeros(n)

    estimates = np.zeros(n)

    for trial in range(t):

        if np.random.rand() < epsilon:

            arm = np.random.randint(n)  # Explore

        else:

            ucb\_values = estimates + np.sqrt((2 \* np.log(trial + 1)) / (counts + 1e-5))  # UCB calculation

            arm = np.argmax(ucb\_values)  # Exploit based on UCB

        reward = simulate\_environment(arm)

        counts[arm] += 1

        estimates[arm] += (reward - estimates[arm]) / counts[arm]

        rewards[trial] = reward

    return rewards

# Run simulations

exploration\_exploitation\_rewards = exploration\_exploitation()

fixed\_exploration\_rewards = fixed\_exploration\_exploitation()

epsilon\_greedy\_rewards = epsilon\_greedy(epsilon=0.1)

epsilon\_greedy\_ucb\_rewards = epsilon\_greedy\_ucb(epsilon=0.1)

# Calculate cumulative rewards for plotting

cumulative\_rewards\_exploration\_exploitation = np.cumsum(exploration\_exploitation\_rewards)

cumulative\_rewards\_fixed\_exploration = np.cumsum(fixed\_exploration\_rewards)

cumulative\_rewards\_epsilon\_greedy = np.cumsum(epsilon\_greedy\_rewards)

cumulative\_rewards\_epsilon\_greedy\_ucb = np.cumsum(epsilon\_greedy\_ucb\_rewards)

# Plotting results

plt.figure(figsize=(12, 8))

plt.plot(cumulative\_rewards\_exploration\_exploitation, label='Exploration & Exploitation', color='blue')

plt.plot(cumulative\_rewards\_fixed\_exploration, label='Fixed Exploration & Exploitation', color='orange')

plt.plot(cumulative\_rewards\_epsilon\_greedy, label='Epsilon-Greedy', color='green')

plt.plot(cumulative\_rewards\_epsilon\_greedy\_ucb, label='Epsilon-Greedy with UCB', color='red')

plt.xlabel('Trials')

plt.ylabel('Cumulative Rewards')

plt.title('Multi-Armed Bandit Strategies Comparison')

plt.legend()

plt.grid()

plt.show()

# Stack the cumulative rewards for each strategy into a 2D array

all\_rewards = np.array([

cumulative\_rewards\_exploration\_exploitation,

cumulative\_rewards\_fixed\_exploration,

cumulative\_rewards\_epsilon\_greedy,

cumulative\_rewards\_epsilon\_greedy\_ucb

])

# Create a heatmap to show how each strategy performs over the trials

plt.figure(figsize=(15, 8))

sns.heatmap(all\_rewards, cmap='YlGnBu', annot=False, xticklabels=50, yticklabels=[

'Exploration & Exploitation', 'Fixed Exploration & Exploitation', 'Epsilon-Greedy', 'Epsilon-Greedy with UCB'])

plt.xlabel('Trials')

plt.ylabel('Strategies')

plt.title('Heatmap of Cumulative Rewards for Different Strategies')

plt.show()

plt.figure(figsize=(20, 8))

# Stacked line plot for cumulative rewards

plt.plot(cumulative\_rewards\_exploration\_exploitation, label='Exploration & Exploitation', color='blue')

plt.plot(cumulative\_rewards\_fixed\_exploration, label='Fixed Exploration & Exploitation', color='orange')

plt.plot(cumulative\_rewards\_epsilon\_greedy, label='Epsilon-Greedy', color='green')

plt.plot(cumulative\_rewards\_epsilon\_greedy\_ucb, label='Epsilon-Greedy with UCB', color='red')

plt.fill\_between(range(len(cumulative\_rewards\_exploration\_exploitation)),

cumulative\_rewards\_exploration\_exploitation, color='blue', alpha=0.2)

plt.fill\_between(range(len(cumulative\_rewards\_fixed\_exploration)),

cumulative\_rewards\_fixed\_exploration, color='orange', alpha=0.2)

plt.fill\_between(range(len(cumulative\_rewards\_epsilon\_greedy)),

cumulative\_rewards\_epsilon\_greedy, color='green', alpha=0.2)

plt.fill\_between(range(len(cumulative\_rewards\_epsilon\_greedy\_ucb)),

cumulative\_rewards\_epsilon\_greedy\_ucb, color='red', alpha=0.2)

plt.xlabel('Trials')

plt.ylabel('Cumulative Rewards')

plt.title('Stacked Cumulative Rewards Comparison')

plt.legend()

plt.grid(True)

plt.show()

**A graph with different colored lines

Description automatically generated**

**A chart with a gradient of blue and green

Description automatically generated**

**A graph with lines and a line

Description automatically generated with medium confidence**

**A graph of a graph

Description automatically generated**

**B.2 Observations and Learning:**

**Effect of Number of Arms (n)**

* **With Fewer Arms (n=3):**
  + Exploration and Exploitation: Quickly identifies the optimal arm due to limited options.
  + Fixed Exploration: Facilitates faster identification of the best arm.
  + Epsilon-Greedy: Generally performs well but may miss optimal rewards without sufficient exploration.
* **With Moderate Arms (n=5):**
  + Performance differences become more pronounced.
  + Epsilon-Greedy: Balances exploration and exploitation, though it may need fine-tuning of ϵ.
  + UCB: Typically outperforms epsilon-greedy due to its adaptive nature.
* **With More Arms (n=10):**
  + Increased complexity leads to higher initial regret as more options are explored.
  + Exploration and Exploitation: May struggle if trials are insufficient for exploration.
  + Epsilon-Greedy and UCB: More effective over time but may exhibit poorer initial performance due to uncertainty.

**Effect of Number of Trials (t)**

* **With Fewer Trials (t=100):**
  + All strategies may fail to converge on the optimal arm, especially with more arms.
  + Results exhibit higher randomness due to limited data.
* **With Moderate Trials (t=500):**
  + Strategies start demonstrating their strengths.
  + Epsilon-Greedy: Begins to perform better as it collects more data.
  + UCB: Shows significant improvement over epsilon-greedy by adapting based on exploration history.
* **With More Trials (t=1000):**
  + Strategies stabilize, allowing for clearer performance comparisons.
  + UCB consistently outperforms other methods due to its effective balance of exploration and exploitation.
  + Cumulative rewards increase significantly, highlighting the benefits of sustained exploration in earlier trials.

**B.3 Conclusion:**

*(****Students must write the conclusive statements as per the actual attainment of individual outcomes listed above and learning/observation noted in section B.2)***

## The analysis of the multi-armed bandit problem reveals that the number of arms and trials significantly impacts the effectiveness of various strategies. With fewer arms, algorithms like epsilon-greedy can quickly identify optimal options, while increasing complexity with more arms and trials necessitates adaptive methods like UCB for improved performance over time.