ID6002W: Online Learning and Reinforcement Learning

Mid Term Exam

Course Instructor: Arun Rajkumar.

Exam Date: 18-06-2023

Prepared By: Aloy Banerjee

Roll Number: CH22M503

(1) State the Doubling trick formally. Explain what problem it solves and show how. (5 points)

ID6002W: Online Learning and Reinforcement Learning
Mid Term Exam
Course Instructor: Arun Rajkumar.
Exam Date: 18-06-2023
Prepared By: Aloy Banerjee
Roll Number: CH22M503

Q.1) Doubling Trick Algorithm :— Algorithm follows below steps,

1. For $m = 1, 2 \dots$
2. Run a new instance of algorithm on the 2^m rounds $t = 2^m, \dots (2^{m+1} - 1)$ with optimal learning rate. (for an algorithm that runs for 2^m steps)

Problem Doubling Trick Solved :— For randomize weighted majority algorithm zero regret bound can be achieve if $\epsilon^* = \sqrt{\frac{\ln d}{T}}$; where, $T \to$ No. of round, $d \to$ No. of expert

Mathematically it state that for weighted majority algorithm, if $\underbrace{T}_{\text{No. of round}} \to \alpha$ then $\underbrace{R_A(T)}_{\text{Algorithm regret}} \to 0$

It actually means that as the number of round increase our algorithm can track the best expert & achieve the zero regret.

But the problem here is, in real life scenario it is impossible to reach $T \to \alpha$ for any practical case.

As the number of round in practical scenario is always finite.

To overcome this issue one approach is to come up with some Infinite Horizon Regret Bound.

To simulate the algorithm for $T \to \alpha$ we can perform below task,

a) Let the algorithm fits T as some finite value, say N.

b) complete the algorithm for first N round & keep track of the loss suffered.

c) Restart the algorith again & run it for same N no of round.

ID6002W: Online Learning and Reinforcement Learning

Mid Term Exam

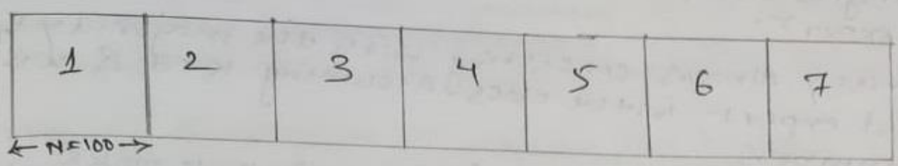
Course Instructor: Arun Rajkumar.

Exam Date: 18-06-2023

Prepared By: Aloy Banerjee

Roll Number: CH22M503

Below is the 7 iteration & each having 100 round as per our example,



Loss suffered in each round is same.

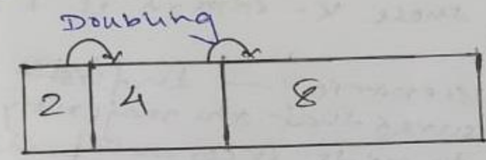
In above diagram we have chousen $N=100$ & restart the algorithm in each 100 round. We have observed the same loss has been suffered as at each block algorithm is restarting.

Problem in this set up is we will never achieve good (improvement) in the suffered loss.

That also means if we make $T \to \infty$, in this way there will be no improvement in regret.

To solve this issue, "Doubling Trick" algorithm come into picture.

If we change the settings in such a way that if we change the no. of iteration round in every steps that means if doubled, diagram will be as follows,



if we use this settings then total regret will be,

$$\sum_{i=0}^{\log T} \sqrt{2^i \log d}$$

[d → No. of expert, T → No. of round]

$$= \sqrt{\log d} \sum_{i=0}^{\log T} 2^{i/2}$$

∵ [As per formula we know, $\sum_{i=1}^{k} a^i = \frac{a^k - 1}{a - 1}$]

Simplifying the code we got, $= \sqrt{\log d} \cdot \sqrt{T}$

Which indicate that loss has suffered in order of $\sqrt{T \log d}$.

Loss suffered $\approx O(\sqrt{T \log d})$

Conclusion:— So, in this settings we will modify the ϵ value at every round after doubling the round & we will get the regret gurantees, which is in the order of $O(\sqrt{T \log d})$

This is called "Doubling Trick".

ID6002W: Online Learning and Reinforcement Learning

Mid Term Exam

Course Instructor: Arun Rajkumar.

Exam Date: 18-06-2023

Prepared By: Aloy Banerjee

Roll Number: CH22M503

(2) Consider the usual online learning protocol with a finite set of d experts. Consider an adversary who is always consistent with the majority of k out of d experts in the class (assume $k < d$ and k is odd). What is the worst case number of mistakes for any algorithm for this problem? Can you think of an algorithm which achieves the same? (10 points)

ID6002W: Online Learning and Reinforcement Learning
Mid Term Exam
Course Instructor: Arun Rajkumar.
Exam Date: 18-06-2023
Prepared By: Aloy Banerjee
Roll Number: CH22M503

Q.2)

According to the problem statement there are finite set of d expert.

Adversery always consisting with the majority of K out of d experts, in the class assuming K < d & K is an odd number.

Main goal of an algorithm (learner) is to make prediction that are as accurate as the best possible expert in hind sight.

As per the problem statement, we can think this problem in broadly 2 way,

a) Either K experts are consistent with adversery.

b) $\frac{K+1}{2}$ experts are consistent with adversery.

If we explain this with some example, lets consider,

if K = 11

-> 11 experts consistent with adversery

-> 6 experts consistent with adversery

if we consider both cases mentioned above,

a) If we go with majority vote algorithm :-

d	K	our decision	Truth	conclusion
64	11			
32		0	1	[Mistake]
16		0	1	[Mistake]

divided by 2

In both the steps, there is a mistake happen, so we are half the expert in each steps are removed.

Since, we know that at least K are consistent, so we have to wait till they get the majority vote.

Now for the explained example, there are 2 mistakes are made.

ID6002W: Online Learning and Reinforcement Learning
Mid Term Exam
Course Instructor: Arun Rajkumar.
Exam Date: 18-06-2023
Prepared By: Aloy Banerjee
Roll Number: CH22M503

similarly,

d	k	No. of mistakes
64	11	2
64	1	6

looking at that table we can observe that, as the value of the k increases, no of mistakes decrease. Since as per problem statement we have to find out the worst number of mistakes, k value has to be decrease. actually k should be minimum.

b) As per the problem statement we are open to debate on any algorithm for worst case mistakes, if we are start picking our expert randomly & adversery force us to make all the output as incorrect then we will end up with making $(d-k)$ mistakes.

And if we consider the worst case $k=1$ then that will become $(d-1)$ mistakes.

We know,

$(d-1)$ mistake $> \log_2(d)$

we can say that if we do random guesses for our output the we can achieve the worst mistakes of $(d-1)$.

Now if we consider the next interpretation, $\frac{k+1}{2}$ experts are consistent with adversary

→ Then, from the previous conclusion we can draw the fact that,

$d - \left(\frac{k+1}{2}\right)$ no. of mistakes will happen for worst case scenario

i.e $(2d-k-1/2)$.

So, in this scenario k can have minimum value of 3 such that worst case mistakes can be $(d-2)$.

ID6002W: Online Learning and Reinforcement Learning
Mid Term Exam
Course Instructor: Arun Rajkumar.
Exam Date: 18-06-2023
Prepared By: Aloy Banerjee
Roll Number: CH22M503

(3) Assume that you want to estimate the probability of a coin coming up heads. Let the true probability be μ.

- Assume $\mu = 0.75$. If you want to make a statement with at least 99% confidence that the estimate $\hat{\mu}$ satisfies $|\mu - \hat{\mu}| \leq 0.1$, how many times would you have to toss the coin? (4 points)
- If the true μ changed from 0.75 to 0.5, how does your answer from the first part change? (2 points)
- If the confidence changed from 99% to 90%, how does your answer from the first part change? (2 points)
- If the tolerance change from 0.1 to 0.01, how does your answer from the first part change? (2 points)

ID6002W: Online Learning and Reinforcement Learning
Mid Term Exam
Course Instructor: Arun Rajkumar.
Exam Date: 18-06-2023
Prepared By: Aloy Banerjee
Roll Number: CH22M503

Q3) Assuming that I want to estimate the probability of a coin coming up as heads.

Let the true probability is μ.

$\hat{\mu}$ is the estimates which satisfy $|\mu - \hat{\mu}| \leq 0.1$

According to the "Hoeffding's Inequality" provides an upper bound on the probability that the sum of bounded independent random variables deviate from its expected value by more than a certain amount.

$$P(|\mu - \hat{\mu}| > \epsilon) \leq 2\exp(-2n\epsilon^2) \quad \cdots (i)$$

Let consider

$$P(|\mu - \hat{\mu}| > \epsilon) = \alpha$$

then equation (i) can be written as,

$$\alpha \leq 2\exp(-2n\epsilon^2)$$

take log in both side of the equation,

$$\Rightarrow \ln\alpha \leq \ln[2\exp(-2n\epsilon^2)]$$

$$\Rightarrow \ln\alpha \leq \ln 2 + (-2n\epsilon^2)$$

$$\Rightarrow \ln\alpha \leq \ln 2 - 2n\epsilon^2$$

PART 1:- $\Rightarrow n \geq \dfrac{\ln(2/\alpha)}{2\epsilon^2} \quad \cdots (ii)$

Now according to the problem statement,

for 99% confidence bound $\alpha = (1 - .99) = 0.01$

& $\epsilon = 0.1$

plotting the value in equation (ii)

$$n \geq \frac{\ln(2/0.01)}{2 \times (0.1)^2}$$

$$\Rightarrow n \geq \frac{5.29832}{0.02}$$

$$\Rightarrow n \geq 264.9 \approx 265$$

$\therefore$ we have to toss the coin for at least 265 times.

ID6002W: Online Learning and Reinforcement Learning

Mid Term Exam

Course Instructor: Arun Rajkumar.

Exam Date: 18-06-2023

Prepared By: Aloy Banerjee

Roll Number: CH22M503

PART 2:-

If true μ changes from 0.75 to 0.5 then there will not be having any impact.

Because if we consider "Hoeffding Inequality" then it is clear that n value does not depend on the true probability.

$$n \geq \frac{\ln(2/\alpha)}{2\epsilon^2}$$; no true probability term exists in this equation.

PART 3:-

As per the equation (ii) from the general equation derived using "Hoeffding Inequality" we know that,

$$n \geq \frac{\ln(2/\alpha)}{2\epsilon^2}$$

As the confidence level changes from 99% to 90%. we can place,

$$\alpha = 0.1 \ \& \ \epsilon = 0.1$$

so, putting this value in the formula we got,

$$n \geq \frac{\ln(2/\alpha)}{2\epsilon^2}$$

$$\Rightarrow n \geq \frac{\ln(2/0.1)}{2 \times (0.1)^2}$$

$$\Rightarrow n \geq \frac{2.9957}{0.02} \approx 149.785 \approx 150$$

$\therefore$ In this settings we have to toss the coin for at least 150 times.

PART 4:-

In this settings tolarance change from 0.1 to 0.01. Then in this case values will be like as follows,

$$\alpha = 0.01 \ \& \ \epsilon = 0.01$$

$$n \geq \frac{\ln(2/\alpha)}{2\epsilon^2}$$

$$\Rightarrow n \geq \frac{\ln(2/0.01)}{2 \times (0.01)^2}$$

$$\Rightarrow n \geq 26491.586 \approx 26492$$

$\therefore$ so in this settings we have to flip the coin at least 26492 times.

ID6002W: Online Learning and Reinforcement Learning

Mid Term Exam

Course Instructor: Arun Rajkumar.

Exam Date: 18-06-2023

Prepared By: Aloy Banerjee

Roll Number: CH22M503

(4) An *epsilon*-greedy strategy for the stochastic multi-armed bandits set up exploits the current best arm with probability $(1-\epsilon)$ and explores with a small probability ϵ. Consider a problem instance with 10 arms where the reward for the i-th $(i = 1, \ldots, 10)$ arm is Beta distributed with parameters $\alpha_i = 5, \beta_i = 5 * i$.

- Implement the *epsilon*-greedy algorithm and compare it with the performance of the UCB and the EXP-3 algorithm. (5 points)
- Comment on your observations about the regret plots obtained in the previous part. (2.5 points)
- If you vary ϵ, how does the regret change? (2.5 points)

ID6002W: Online Learning and Reinforcement Learning
Mid Term Exam
Course Instructor: Arun Rajkumar.

Exam Date: 18-06-2023

Prepared By: Aloy Banerjee

Roll Number: CH22M503

Screenshot:

Part 1 - Implement the epsilon-greedy algorithm and compare it with the performance of the UCB and the EXP-3 algorithm. (5 points)

Importing Library

```
import numpy as np
import matplotlib as mat
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

Common Variable

```
# Setting seed to control randomness for reproducibility
seed = 48

# Initializing random number generator with the defined seed
np.random.seed(seed)

# Defining common variables for running the algorithm
num_arms = 10  # Number of arms/options in the multi-armed bandit problem
num_iterations = 1000  # Number of iterations to run the algorithm

# Parameters of the Beta distributions for each arm's reward
# Success parameters (alpha) for each arm's Beta distribution, all set to 5
success_params = np.ones(num_arms) * 5
# Failure parameters (beta) for each arm's Beta distribution, increasing by 5 for each arm
failure_params = np.arange(5, 5*(num_arms+1), 5)

# Reshape failure_params to match the shape of success_params
# Ensuring the failure_params array has same shape as success_params
failure_params = np.resize(failure_params, success_params.shape)
```

ID6002W: Online Learning and Reinforcement Learning

Mid Term Exam

Course Instructor: Arun Rajkumar.

Exam Date: 18-06-2023

Prepared By: Aloy Banerjee

Roll Number: CH22M503

```python
# Epsilon greedy Algorithm
def epsilon_greedy_policy(epsilon_val, num_options, num_iterations, success_params, failure_params):

    """
    Implements the epsilon-greedy algorithm for a multi-armed bandit problem.

    Args:
        epsilon_val (float): The probability of choosing a random option (exploration).
        num_options (int): The number of options, or "arms," of the bandit.
        num_iterations (int): The number of iterations to run the algorithm.
        success_params (array_like): Parameters of the Beta distribution for success for each option.
        failure_params (array_like): Parameters of the Beta distribution for failure for each option.

    Returns:
        np.array: The cumulative regret after each time step.

    This function operates over a specified number of iterations. At each time step,
    it chooses the option with the highest average reward with probability (1-epsilon),
    and chooses a random option with probability epsilon. The function calculates the
    "regret" of each option as the difference in success probability from the optimal
    option. The function returns the cumulative regret after each time step.
    """

    # Initialize cumulative rewards and number of times each option is chosen
    total_rewards = np.zeros(num_options)
    option_uses = np.zeros(num_options)

    # Initialize rewards and regret lists for tracking over time
    reward_tracker = []
    regret_tracker = []

    # Determine the "best" option under current understanding
    optimal_option = np.argmax((success_params / (success_params + failure_params)))

    # Begin the main loop for the specified number of time steps
    for time_step in range(num_iterations):
        # Choose the option with highest average reward most of the time (probability 1-epsilon)
        if np.random.random() < (1 - epsilon_val):
            # Add small value to prevent division by zero
            selected_option = np.argmax(total_rewards / (option_uses + 1e-6))
        else:  # but choose randomly some of the time (probability epsilon)
            selected_option = np.random.randint(num_options)

        # Generate reward based on a Beta distribution (specific to each option)
        current_reward = np.random.beta(success_params[selected_option], failure_params[selected_option])

        # Update rewards and usage for the selected option
        total_rewards[selected_option] += current_reward
        option_uses[selected_option] += 1

        # Record current reward and regret (difference in success probability from optimal option)
        reward_tracker.append(current_reward)
        regret_tracker.append(abs(success_params[optimal_option] / (success_params[optimal_option]
                                                                  + failure_params[optimal_option])
                                  - success_params[selected_option] / (success_params[selected_option]
                                                                     + failure_params[selected_option])))

    # Return cumulative regret over time
    return np.cumsum(regret_tracker)
```

ID6002W: Online Learning and Reinforcement Learning

Mid Term Exam

Course Instructor: Arun Rajkumar.

Exam Date: 18-06-2023

Prepared By: Aloy Banerjee

Roll Number: CH22M503

```python
# UCB algorithm
def UCB_algorithm(num_options, num_iterations, success_params, failure_params):
    """
    Implements the Upper Confidence Bound (UCB) algorithm for a multi-armed bandit problem.

    Args:
        num_options (int): The number of options, or "arms," of the bandit.
        num_iterations (int): The number of iterations to run the algorithm.
        success_params (array_like): Parameters of the Beta distribution for success for each option.
        failure_params (array_like): Parameters of the Beta distribution for failure for each option.

    Returns:
        np.array: The cumulative regret after each time step.

    The function operates over a specified number of iterations. At each time step,
    it calculates the UCB value for each option, and chooses the option with the
    highest UCB value. The UCB value for an option is its average reward plus an
    uncertainty term, which decreases as that option is chosen more frequently.

    The function calculates the "regret" of each option as the difference in success
    probability from the optimal option. The function returns the cumulative regret
    after each time step.
    """

    # Initialize cumulative rewards and number of times each option is chosen
    total_rewards = np.zeros(num_options)
    option_uses = np.ones(num_options)

    # Initialize rewards and regret lists for tracking over time
    reward_list = []
    regret_list = []

    # Determine the "best" option under current understanding
    optimal_option = np.argmax((success_params / (success_params + failure_params)))

    # Begin the main loop for the specified number of time steps, starting from num_options
    # (as each option needs to be tried once)
    for time_step in range(num_options, num_iterations):
        # Calculate UCB values for each option
        ucb_values = total_rewards / option_uses + np.sqrt(2 * np.log(time_step) / option_uses)
        # Choose the option that currently has the highest UCB value
        selected_option = np.argmax(ucb_values)

        # Generate reward based on a Beta distribution (specific to each option)
        current_reward = np.random.beta(success_params[selected_option], failure_params[selected_option])

        # Update rewards and usage for the selected option
        total_rewards[selected_option] += current_reward
        option_uses[selected_option] += 1

        # Record current reward and regret (difference in success probability from optimal option)
        reward_list.append(current_reward)
        regret_list.append(abs(success_params[optimal_option] / (success_params[optimal_option]
                                                                  + failure_params[optimal_option])
                               - success_params[selected_option] / (success_params[selected_option]
                                                                     + failure_params[selected_option])))

    # Return cumulative regret over time
    return np.cumsum(regret_list)
```

ID6002W: Online Learning and Reinforcement Learning
Mid Term Exam
Course Instructor: Arun Rajkumar.
Exam Date: 18-06-2023

Prepared By: Aloy Banerjee
Roll Number: CH22M503


```python
# EXP-3 algorithm
def EXP3_algorithm(num_options, num_iterations, success_params, failure_params, learning_rate):
    """
    Implements the Exponential-weight algorithm for Exploration and Exploitation (EXP3) for a multi-armed bandit problem.

    Args:
        num_options (int): The number of options, or "arms," of the bandit.
        num_iterations (int): The number of iterations to run the algorithm.
        success_params (array_like): Parameters of the Beta distribution for success for each option.
        failure_params (array_like): Parameters of the Beta distribution for failure for each option.
        learning_rate (float): Learning rate parameter for the EXP3 algorithm.

    Returns:
        np.array: The cumulative regret after each time step.

    The function operates over a specified number of iterations. At each time step,
    it calculates the probabilities for each option based on their weights and selects
    an option randomly according to the calculated probabilities.

    The function calculates the "regret" of each option as the difference in success
    probability from the optimal option. The function returns the cumulative regret
    after each time step.
    """
    # Initialize cumulative rewards and weights for each option
    total_rewards = np.zeros(num_options)
    option_weights = np.ones(num_options)

    # Initialize rewards and regret lists for tracking over time
    reward_records = []
    regret_records = []

    # Determine the "best" option under current understanding
    optimal_option = np.argmax((success_params / (success_params + failure_params)))

    # Begin the main loop for the specified number of time steps
    for time_step in range(num_iterations):
        # Calculate probabilities for each option
        option_probs = (1 - learning_rate) * (option_weights / np.sum(option_weights)) + learning_rate / num_options
        # Select an option randomly according to the calculated probabilities
        selected_option = np.random.choice(num_options, p=option_probs)

        # Generate reward based on a Beta distribution (specific to each option)
        current_reward = np.random.beta(success_params[selected_option], failure_params[selected_option])

        # Update the total reward for the selected option
        total_rewards[selected_option] += current_reward

        # Estimate the reward and update the weight of the selected option
        estimated_reward = total_rewards[selected_option] / option_probs[selected_option]
        option_weights[selected_option] *= np.exp(learning_rate * estimated_reward / num_options)

        # Normalize weights to prevent underflow
        option_weights /= np.sum(option_weights)

        # Record current reward and regret (difference in success probability from optimal option)
        reward_records.append(current_reward)
        regret_records.append(abs(success_params[optimal_option] / (success_params[optimal_option] + failure_params[optimal

    # Return cumulative regret over time
    return np.cumsum(regret_records)
```

ID6002W: Online Learning and Reinforcement Learning

Mid Term Exam

Course Instructor: Arun Rajkumar.

Exam Date: 18-06-2023

Prepared By: Aloy Banerjee

Roll Number: CH22M503

```python
def plot_cumulative_regret(epsilon_greedy_cumulative_regret, ucb_cumulative_regret, exp3_cumulative_regret, epsilon_values,
    """
    Plots the cumulative regret for Epsilon-Greedy, UCB, and EXP3 algorithms.

    Args:
        epsilon_greedy_cumulative_regret (array_like): Cumulative regret values from the Epsilon-Greedy algorithm.
        ucb_cumulative_regret (array_like): Cumulative regret values from the UCB algorithm.
        exp3_cumulative_regret (array_like): Cumulative regret values from the EXP3 algorithm.
        epsilon_values (float): Epsilon value used in the Epsilon-Greedy algorithm.
        learning_params (float): Learning parameter used in the EXP3 algorithm.

    The function creates a line plot for each set of regret values, using different colors
    for each algorithm. It sets labels for the x-axis and y-axis, a title for the plot, and
    a legend. It also includes grid lines for easier viewing of the data points.
    """
    # Create a colormap using the 'cool' colormap from the matplotlib library
    cmap = mat.colormaps['cool']
    # Generate three colors from the colormap. These colors are evenly spaced
    # from the beginning (0) to the end (1) of the colormap.
    colors = cmap(np.linspace(0, 1, 3))
    # Set the figure size for the plot
    plt.figure(figsize=(10,7))
    # Plot the cumulative regret for the Epsilon-Greedy algorithm.
    # The color for this line is the first color from the colormap.
    plt.plot(epsilon_greedy_cumulative_regret, label="Epsilon-Greedy", color=colors[0])
    # Plot the cumulative regret for the UCB algorithm.
    # The color for this line is the second color from the colormap.
    plt.plot(ucb_cumulative_regret, label="UCB", color=colors[1])
    # Plot the cumulative regret for the EXP-3 algorithm.
    # The color for this line is the third color from the colormap.
    plt.plot(exp3_cumulative_regret, label="EXP-3", color=colors[2])
    # Set the label for the x-axis
    plt.xlabel("No. of Rounds", fontsize=14)
    # Set the label for the y-axis
    plt.ylabel("Cumulative Regret", fontsize=14)
    # Set the title for the plot
    plt.title(f"Comparison of Epsilon-Greedy, UCB, and EXP-3 algorithms for epsilon value {epsilon_values} & learning param
    # Display the legend
    plt.legend(fontsize=12)
    # Display grid lines
    plt.grid(True, linestyle='--', alpha=0.6)
    # Show the plot
    plt.show()
```

ID6002W: Online Learning and Reinforcement Learning
Mid Term Exam
Course Instructor: Arun Rajkumar.
Exam Date: 18-06-2023
Prepared By: Aloy Banerjee
Roll Number: CH22M503

```python
def process_results(df, results, algorithm_name, epsilon_val, learning_param):
    """
    Process the results of a multi-armed bandit algorithm, recording data for each set of parameters.

    Args:
        df (pandas.DataFrame): The DataFrame to which the results will be appended.
        results (list): A list of tuples containing the parameters and regret values for each run of the algorithm.
        algorithm_name (str): The name of the algorithm used.

        Each tuple in `results` should contain three elements:
        1. epsilon_val (float): The epsilon value used in the algorithm run.
        2. learning_param (float): The learning parameter used in the algorithm run.
        3. regrets (list): A list of regret values from the algorithm run.

    The function identifies the maximum and minimum regret in each run and their respective iterations,
    creates a new DataFrame with these details, and appends it to the provided DataFrame `df`.

    Returns:
        df_analyser (pandas.DataFrame): The original DataFrame, updated with the new data.
    """
    # Iterate through each tuple in the results list
    for epsilon_val, learning_param, regrets in results:
        # Find the maximum regret value and its corresponding iteration number
        max_regret_iter, max_regret = max(enumerate(regrets), key=lambda x: x[1])
        # Find the minimum regret value and its corresponding iteration number
        min_regret_iter, min_regret = min(enumerate(regrets), key=lambda x: x[1])

        # Create a new DataFrame to store the results of the current tuple
        new_data = pd.DataFrame({
            'Algorithm': [algorithm_name], # The algorithm name
            'Epsilon': [epsilon_val], # The epsilon value used
            'Learning_Rate': [learning_param], # The learning parameter used
            'Max_Regret': [max_regret], # The maximum regret value
            'Max_Regret_Iter': [max_regret_iter + 1], # The iteration number for the maximum regret value
            'Min_Regret': [min_regret], # The minimum regret value
            'Min_Regret_Iter': [min_regret_iter + 1] # The iteration number for the minimum regret value
        })

        # Append the new data to the existing DataFrame
        df_analyser = pd.concat([df, new_data], ignore_index=True)

    # Return the updated DataFrame
    return df_analyser
```

ID6002W: Online Learning and Reinforcement Learning

Mid Term Exam

Course Instructor: Arun Rajkumar.

Exam Date: 18-06-2023

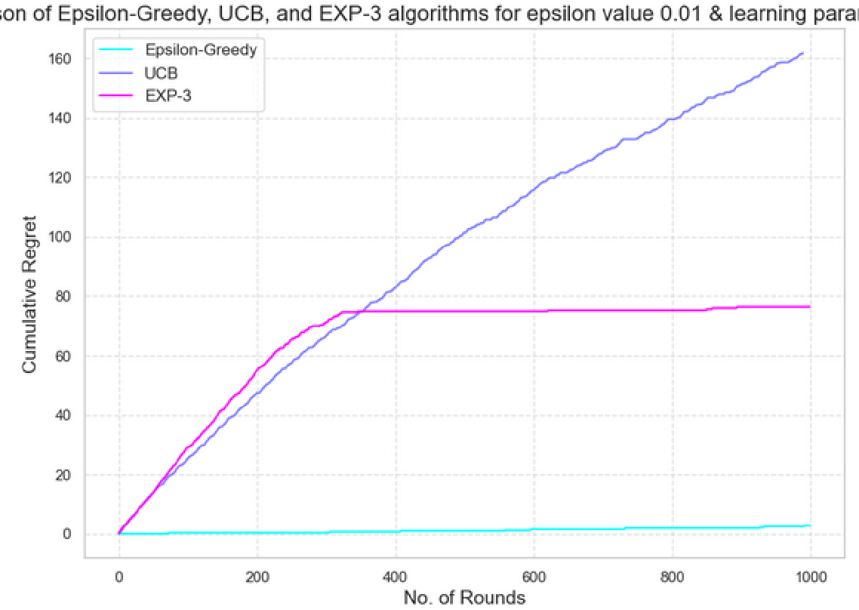
Prepared By: Aloy Banerjee

Roll Number: CH22M503

Execution of the algorithm ¶

```python
# The probability used in the epsilon-greedy algorithm to determine
# whether to explore (choose a random arm) or exploit (choose the best arm so far).
epsilon_values = [0.01]
learning_params = [0.01]

# Prepare empty lists to store results
epsilon_greedy_cumulative_regret, ucb_cumulative_regret, exp3_cumulative_regret = [], [], []

# Prepare empty lists to store results
epsilon_greedy_results, ucb_results, exp3_results = [], [], []


# Run simulations for each combination of epsilon and learning parameters
for epsilon_val in epsilon_values:
    for learning_param in learning_params:
        epsilon_greedy_cumulative_regret = epsilon_greedy_policy(epsilon_val, num_arms, num_iterations, success_params, fai
        ucb_cumulative_regret = UCB_algorithm(num_arms, num_iterations, success_params, failure_params)
        exp3_cumulative_regret = EXP3_algorithm(num_arms, num_iterations, success_params, failure_params, learning_param)
        plot_cumulative_regret(epsilon_greedy_cumulative_regret, ucb_cumulative_regret, exp3_cumulative_regret, epsilon_val
        epsilon_greedy_results.append((epsilon_val, learning_param, epsilon_greedy_cumulative_regret))
        ucb_results.append((epsilon_val, learning_param, ucb_cumulative_regret))
        exp3_results.append((epsilon_val, learning_param, exp3_cumulative_regret))

```

Comparison of Epsilon-Greedy, UCB, and EXP-3 algorithms for epsilon value 0.01 & learning parameter value 0.01



ID6002W: Online Learning and Reinforcement Learning
Mid Term Exam
Course Instructor: Arun Rajkumar.
Exam Date: 18-06-2023
Prepared By: Aloy Banerjee
Roll Number: CH22M503

Loading the data in dataframe for analysis ¶

```
# Create an empty DataFrame
df_Analyser = pd.DataFrame(columns=['Algorithm', 'Epsilon', 'Learning_Rate', 'Max_Regret',
                                    'Max_Regret_Iter', 'Min_Regret', 'Min_Regret_Iter'])

# Process results for each algorithm and add to the DataFrame
df_Analyser = process_results(df_Analyser, epsilon_greedy_results, 'Epsilon-Greedy', epsilon_values[0],
                              learning_params[0])
df_Analyser = process_results(df_Analyser, ucb_results, 'UCB', epsilon_values[0], learning_params[0])
df_Analyser = process_results(df_Analyser, exp3_results, 'Exp 3', epsilon_values[0], learning_params[0])
df_Analyser_sorted = df_Analyser.sort_values(['Epsilon', 'Learning_Rate'], ascending=[True, True])
```

Viewed the sorted dataframe containing the information about the regret for different algorithm for selected Epsilon and Learning rate

```
df_Analyser_sorted
```

87]:

	Algorithm	Epsilon	Learning_Rate	Max_Regret	Max_Regret_Iter	Min_Regret	Min_Regret_Iter
0	Epsilon-Greedy	0.01	0.01	2.842063	993	0.0	1
1	UCB	0.01	0.01	161.713203	990	0.0	1
2	Exp 3	0.01	0.01	76.372691	895	0.3	1

Part 2 : Comment on your observations about the regret plots obtained in the previous part. (2.5 points)

Conclusion of Part 1 :

- Looking at the plot for three algorithm reveals that the value of cumulative regret over the number of rounds,
  - for **UCB** : Remain in increasing trend.
  - for **Exp-3** : Initially the regret value is increasing but eventually it flattened out after certain round.
  - for **Epsilon Greedy** : On the contrary cumulative regret remain very close to zero and change very slowly.
- We are getting same indication from the table displayed in the last cell also.

ID6002W: Online Learning and Reinforcement Learning
Mid Term Exam
Course Instructor: Arun Rajkumar.
Exam Date: 18-06-2023
Prepared By: Aloy Banerjee
Roll Number: CH22M503

Part 3 : Execution of the algorithm for different values of ϵ

```python
seed = 20
np.random.seed(seed)
num_iterations = 1000
# Different epsilon values to explore
epsilon_values_to_explore = [0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5,
                             0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95]

# List to store the cumulative regrets for each epsilon value
cumulative_regrets_for_each_epsilon = []

# Run the epsilon-greedy policy for each epsilon value and store the cumulative regrets
for epsilon in epsilon_values_to_explore:
    cumulative_regret = epsilon_greedy_policy(epsilon, num_arms, num_iterations, success_params, failure_params)
    cumulative_regrets_for_each_epsilon.append(cumulative_regret)

# Set the style and context of the plot to make it look more attractive
sns.set(style='whitegrid', context='notebook')

# Set the color palette for the plot
colors = [mat.colormaps['tab20'](iloop) for iloop in range(20)]

# Create a larger figure
plt.figure(figsize=(12, 8))

# Loop over epsilon values
for i, epsilon in enumerate(epsilon_values_to_explore):
    # Plot the cumulative regret for each epsilon, with a different color and line style for each one
    plt.plot(cumulative_regrets_for_each_epsilon[i], color=colors[i], linestyle='-', label=f"Epsilon = {epsilon}")

# Set labels and title with larger fonts
plt.xlabel("No. of Rounds", fontsize=14)
plt.ylabel("Cumulative Regret", fontsize=14)
plt.title("Epsilon-Greedy Algorithm with Regret and Various Epsilon Values", fontsize=16)

# Display the legend and increase its font size
plt.legend(fontsize=12)

# Add a slight grid for better readability
plt.grid(True, linestyle='--', alpha=0.6)

# Show the plot
plt.show()
```

ID6002W: Online Learning and Reinforcement Learning

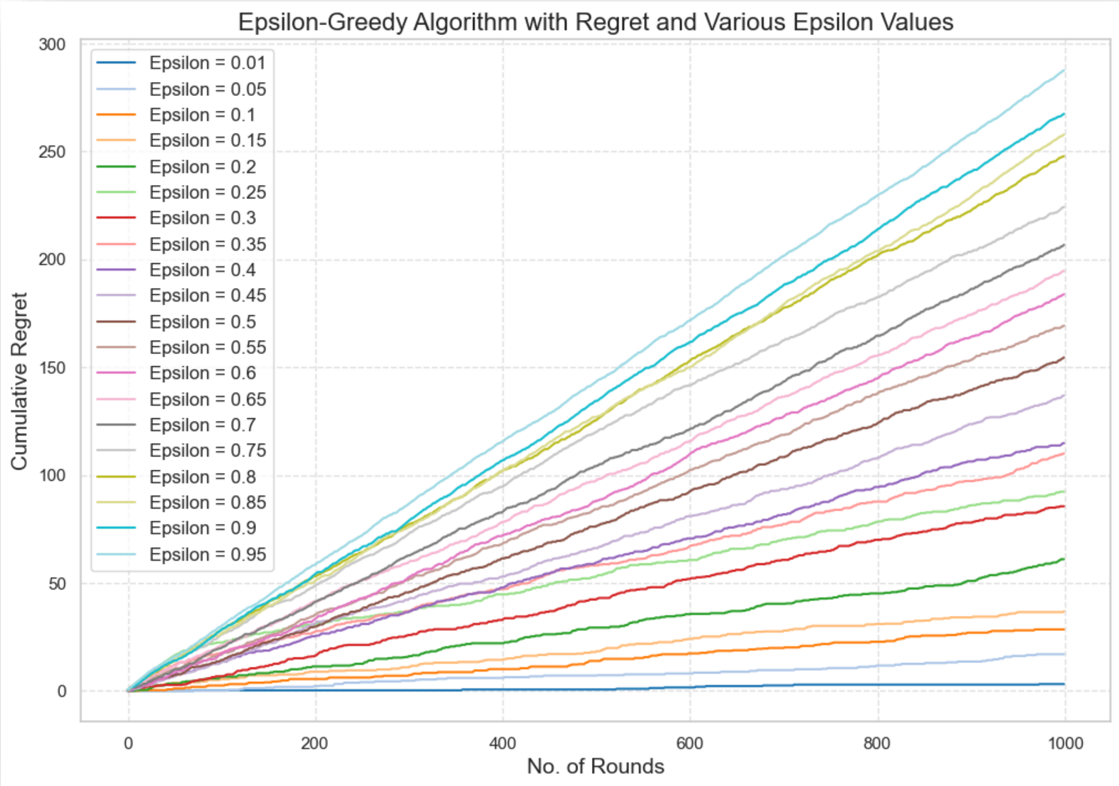
Mid Term Exam

Course Instructor: Arun Rajkumar.

Exam Date: 18-06-2023

Prepared By: Aloy Banerjee

Roll Number: CH22M503



Conclusion Part 3 :

- Various epsilon values (0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95) demonstrate various regret patterns in the Epsilon-Greedy algorithm. In terms of minimizing regrets and observing that the regret is growing its slope as and when we are raising the epsilon value over the number of rounds, smaller epsilon values (for example, 0.01) tend to perform better than bigger epsilon values (for example, 0.95). This suggests that a more experimental strategy (lower epsilon) can eventually result in greater overall performance.

ID6002W: Online Learning and Reinforcement Learning
Mid Term Exam
Course Instructor: Arun Rajkumar.
Exam Date: 18-06-2023

Prepared By: Aloy Banerjee
Roll Number: CH22M503

Code Text:

Part 1 - Implement the epsilon-greedy algorithm and compare it with the performance of the UCB and the EXP-3 algorithm. (5 points)

Importing Library

```
import numpy as np
import matplotlib as mat
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

Common Variable

```
# Setting seed to control randomness for reproducibility
seed = 48

# Initializing random number generator with the defined seed
np.random.seed(seed)

# Defining common variables for running the algorithm
num_arms = 10  # Number of arms/options in the multi-armed bandit problem
num_iterations = 1000  # Number of iterations to run the algorithm

# Parameters of the Beta distributions for each arm's reward
# Success parameters (alpha) for each arm's Beta distribution, all set to
5
success_params = np.ones(num_arms) * 5
# Failure parameters (beta) for each arm's Beta distribution, increasing
by 5 for each arm
failure_params = np.arange(5, 5*(num_arms+1), 5)

# Reshape failure_params to match the shape of success_params
# Ensuring the failure_params array has same shape as success_params
failure_params = np.resize(failure_params, success_params.shape)
```

Common Method

ID6002W: Online Learning and Reinforcement Learning
Mid Term Exam
Course Instructor: Arun Rajkumar.
Exam Date: 18-06-2023
Prepared By: Aloy Banerjee
Roll Number: CH22M503

```
# Epsilon greedy Algorithm
def epsilon_greedy_policy(epsilon_val, num_options, num_iterations,
success_params, failure_params):

    """
    Implements the epsilon-greedy algorithm for a multi-armed bandit
problem.

    Args:
        epsilon_val (float): The probability of choosing a random option
(exploration).
        num_options (int): The number of options, or "arms," of the
bandit.
        num_iterations (int): The number of iterations to run the
algorithm.
        success_params (array_like): Parameters of the Beta distribution
for success for each option.
        failure_params (array_like): Parameters of the Beta distribution
for failure for each option.

    Returns:
        np.array: The cumulative regret after each time step.

    This function operates over a specified number of iterations. At each
time step,
    it chooses the option with the highest average reward with probability
(1-epsilon),
    and chooses a random option with probability epsilon. The function
calculates the
    "regret" of each option as the difference in success probability from
the optimal
    option. The function returns the cumulative regret after each time
step.
    """

    # Initialize cumulative rewards and number of times each option is
chosen
    total_rewards = np.zeros(num_options)
    option_uses = np.zeros(num_options)

    # Initialize rewards and regret lists for tracking over time
    reward_tracker = []
    regret_tracker = []
```

ID6002W: Online Learning and Reinforcement Learning

Mid Term Exam

Course Instructor: Arun Rajkumar.

Exam Date: 18-06-2023

Prepared By: Aloy Banerjee

Roll Number: CH22M503

```
    # Determine the "best" option under current understanding
    optimal_option = np.argmax((success_params / (success_params +
failure_params)))

    # Begin the main loop for the specified number of time steps
    for time_step in range(num_iterations):
        # Choose the option with highest average reward most of the time
(probability 1-epsilon)
        if np.random.random() < (1 - epsilon_val):
            # Add small value to prevent division by zero
            selected_option = np.argmax(total_rewards / (option_uses + 1e-
6))
        else:  # but choose randomly some of the time (probability
epsilon)
            selected_option = np.random.randint(num_options)

        # Generate reward based on a Beta distribution (specific to each
option)
        current_reward = np.random.beta(success_params[selected_option],
failure_params[selected_option])

        # Update rewards and usage for the selected option
        total_rewards[selected_option] += current_reward
        option_uses[selected_option] += 1

        # Record current reward and regret (difference in success
probability from optimal option)
        reward_tracker.append(current_reward)
        regret_tracker.append(abs(success_params[optimal_option] /
(success_params[optimal_option] + failure_params[optimal_option]) -
success_params[selected_option] / (success_params[selected_option] +
failure_params[selected_option])))

    # Return cumulative regret over time
    return np.cumsum(regret_tracker)


# UCB algorithm
def UCB_algorithm(num_options, num_iterations, success_params,
failure_params):
    """
    Implements the Upper Confidence Bound (UCB) algorithm for a multi-
armed bandit problem.
```

ID6002W: Online Learning and Reinforcement Learning
Mid Term Exam
Course Instructor: Arun Rajkumar.
Exam Date: 18-06-2023
Prepared By: Aloy Banerjee
Roll Number: CH22M503

```
    Args:
        num_options (int): The number of options, or "arms," of the
bandit.
        num_iterations (int): The number of iterations to run the
algorithm.
        success_params (array_like): Parameters of the Beta distribution
for success for each option.
        failure_params (array_like): Parameters of the Beta distribution
for failure for each option.

    Returns:
        np.array: The cumulative regret after each time step.

    The function operates over a specified number of iterations. At each
time step,
    it calculates the UCB value for each option, and chooses the option
with the
    highest UCB value. The UCB value for an option is its average reward
plus an
    uncertainty term, which decreases as that option is chosen more
frequently.

    The function calculates the "regret" of each option as the difference
in success
    probability from the optimal option. The function returns the
cumulative regret
    after each time step.
    """

    # Initialize cumulative rewards and number of times each option is
chosen
    total_rewards = np.zeros(num_options)
    option_uses = np.ones(num_options)

    # Initialize rewards and regret lists for tracking over time
    reward_list = []
    regret_list = []

    # Determine the "best" option under current understanding
    optimal_option = np.argmax((success_params / (success_params +
failure_params)))

    # Begin the main loop for the specified number of time steps, starting
from num_options
```

ID6002W: Online Learning and Reinforcement Learning
Mid Term Exam
Course Instructor: Arun Rajkumar.
Exam Date: 18-06-2023
Prepared By: Aloy Banerjee
Roll Number: CH22M503

```
    # (as each option needs to be tried once)
    for time_step in range(num_options, num_iterations):
        # Calculate UCB values for each option
        ucb_values = total_rewards / option_uses + np.sqrt(2 *
np.log(time_step) / option_uses)
        # Choose the option that currently has the highest UCB value
        selected_option = np.argmax(ucb_values)

        # Generate reward based on a Beta distribution (specific to each
option)
        current_reward = np.random.beta(success_params[selected_option],
failure_params[selected_option])

        # Update rewards and usage for the selected option
        total_rewards[selected_option] += current_reward
        option_uses[selected_option] += 1

        # Record current reward and regret (difference in success
probability from optimal option)
        reward_list.append(current_reward)
        regret_list.append(abs(success_params[optimal_option] /
(success_params[optimal_option] + failure_params[optimal_option]) -
success_params[selected_option] / (success_params[selected_option] +
failure_params[selected_option])))

    # Return cumulative regret over time
    return np.cumsum(regret_list)

# EXP-3 algorithm
def EXP3_algorithm(num_options, num_iterations, success_params,
failure_params, learning_rate):
    """
    Implements the Exponential-weight algorithm for Exploration and
Exploitation (EXP3) for a multi-armed bandit problem.

    Args:
        num_options (int): The number of options, or "arms," of the
bandit.
        num_iterations (int): The number of iterations to run the
algorithm.
        success_params (array_like): Parameters of the Beta distribution
for success for each option.
        failure_params (array_like): Parameters of the Beta distribution
for failure for each option.
```

```
        learning_rate (float): Learning rate parameter for the EXP3
algorithm.

    Returns:
        np.array: The cumulative regret after each time step.

    The function operates over a specified number of iterations. At each
time step,
    it calculates the probabilities for each option based on their weights
and selects
    an option randomly according to the calculated probabilities.

    The function calculates the "regret" of each option as the difference
in success
    probability from the optimal option. The function returns the
cumulative regret
    after each time step.
    """
    # Initialize cumulative rewards and weights for each option
    total_rewards = np.zeros(num_options)
    option_weights = np.ones(num_options)

    # Initialize rewards and regret lists for tracking over time
    reward_records = []
    regret_records = []

    # Determine the "best" option under current understanding
    optimal_option = np.argmax((success_params / (success_params +
failure_params)))

    # Begin the main loop for the specified number of time steps
    for time_step in range(num_iterations):
        # Calculate probabilities for each option
        option_probs = (1 - learning_rate) * (option_weights /
np.sum(option_weights)) + learning_rate / num_options
        # Select an option randomly according to the calculated
probabilities
        selected_option = np.random.choice(num_options, p=option_probs)

        # Generate reward based on a Beta distribution (specific to each
option)
        current_reward = np.random.beta(success_params[selected_option],
failure_params[selected_option])
```

```
        # Update the total reward for the selected option
        total_rewards[selected_option] += current_reward

        # Estimate the reward and update the weight of the selected option
        estimated_reward = total_rewards[selected_option] /
option_probs[selected_option]
        option_weights[selected_option] *= np.exp(learning_rate *
estimated_reward / num_options)

        # Normalize weights to prevent underflow
        option_weights /= np.sum(option_weights)

        # Record current reward and regret (difference in success
probability from optimal option)
        reward_records.append(current_reward)
        regret_records.append(abs(success_params[optimal_option] /
(success_params[optimal_option] + failure_params[optimal_option]) -
success_params[selected_option] / (success_params[selected_option] +
failure_params[selected_option])))

    # Return cumulative regret over time
    return np.cumsum(regret_records)


def plot_cumulative_regret(epsilon_greedy_cumulative_regret,
ucb_cumulative_regret, exp3_cumulative_regret, epsilon_values,
learning_params):
    """
    Plots the cumulative regret for Epsilon-Greedy, UCB, and EXP3
algorithms.

    Args:
        epsilon_greedy_cumulative_regret (array_like): Cumulative regret
values from the Epsilon-Greedy algorithm.
        ucb_cumulative_regret (array_like): Cumulative regret values from
the UCB algorithm.
        exp3_cumulative_regret (array_like): Cumulative regret values from
the EXP3 algorithm.
        epsilon_values (float): Epsilon value used in the Epsilon-Greedy
algorithm.
        learning_params (float): Learning parameter used in the EXP3
algorithm.

    The function creates a line plot for each set of regret values, using
```

```
different colors
    for each algorithm. It sets labels for the x-axis and y-axis, a title
for the plot, and
    a legend. It also includes grid lines for easier viewing of the data
points.
    """
    # Create a colormap using the 'cool' colormap from the matplotlib
library
    cmap = mat.colormaps['cool']
    # Generate three colors from the colormap. These colors are evenly
spaced
    # from the beginning (0) to the end (1) of the colormap.
    colors = cmap(np.linspace(0, 1, 3))
    # Set the figure size for the plot
    plt.figure(figsize=(10,7))
    # Plot the cumulative regret for the Epsilon-Greedy algorithm.
    # The color for this line is the first color from the colormap.
    plt.plot(epsilon_greedy_cumulative_regret, label="Epsilon-Greedy",
color=colors[0])
    # Plot the cumulative regret for the UCB algorithm.
    # The color for this line is the second color from the colormap.
    plt.plot(ucb_cumulative_regret, label="UCB", color=colors[1])
    # Plot the cumulative regret for the EXP-3 algorithm.
    # The color for this line is the third color from the colormap.
    plt.plot(exp3_cumulative_regret, label="EXP-3", color=colors[2])
    # Set the label for the x-axis
    plt.xlabel("No. of Rounds", fontsize=14)
    # Set the label for the y-axis
    plt.ylabel("Cumulative Regret", fontsize=14)
    # Set the title for the plot
    plt.title(f"Comparison of Epsilon-Greedy, UCB, and EXP-3 algorithms
for epsilon value {epsilon_values} & learning parameter value
{learning_params}", fontsize=16)
    # Display the legend
    plt.legend(fontsize=12)
    # Display grid lines
    plt.grid(True, linestyle='--', alpha=0.6)
    # Show the plot
    plt.show()


def process_results(df, results, algorithm_name, epsilon_val,
learning_param):
    """
```

ID6002W: Online Learning and Reinforcement Learning
Mid Term Exam
Course Instructor: Arun Rajkumar.
Exam Date: 18-06-2023
Prepared By: Aloy Banerjee
Roll Number: CH22M503

```
    Process the results of a multi-armed bandit algorithm, recording data
for each set of parameters.

    Args:
        df (pandas.DataFrame): The DataFrame to which the results will be
appended.
        results (list): A list of tuples containing the parameters and
regret values for each run of the algorithm.
        algorithm_name (str): The name of the algorithm used.

        Each tuple in `results` should contain three elements:
        1. epsilon_val (float): The epsilon value used in the algorithm
run.
        2. learning_param (float): The learning parameter used in the
algorithm run.
        3. regrets (list): A list of regret values from the algorithm run.

    The function identifies the maximum and minimum regret in each run and
their respective iterations,
    creates a new DataFrame with these details, and appends it to the
provided DataFrame `df`.

    Returns:
        df_analyser (pandas.DataFrame): The original DataFrame, updated
with the new data.
    """
    # Iterate through each tuple in the results list
    for epsilon_val, learning_param, regrets in results:
        # Find the maximum regret value and its corresponding iteration
number
        max_regret_iter, max_regret = max(enumerate(regrets), key=lambda
x: x[1])
        # Find the minimum regret value and its corresponding iteration
number
        min_regret_iter, min_regret = min(enumerate(regrets), key=lambda
x: x[1])

        # Create a new DataFrame to store the results of the current tuple
        new_data = pd.DataFrame({
            'Algorithm': [algorithm_name], # The algorithm name
            'Epsilon': [epsilon_val], # The epsilon value used
            'Learning_Rate': [learning_param], # The learning parameter
used
            'Max_Regret': [max_regret], # The maximum regret value
```

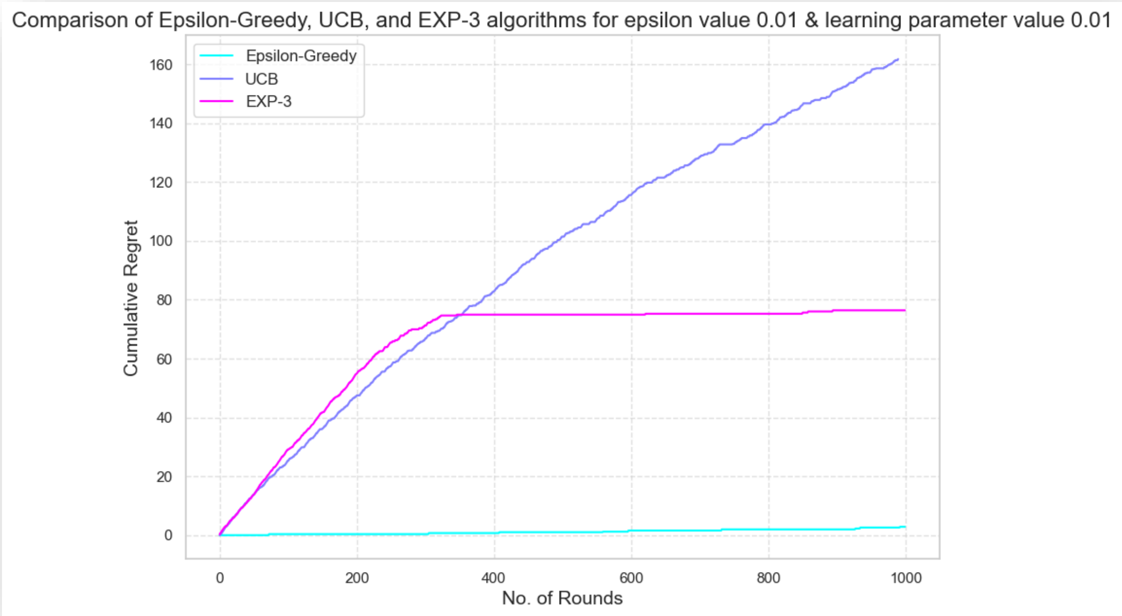
```
            'Max_Regret_Iter': [max_regret_iter + 1], # The iteration
number for the maximum regret value
            'Min_Regret': [min_regret], # The minimum regret value
            'Min_Regret_Iter': [min_regret_iter + 1] # The iteration
number for the minimum regret value
        })

        # Append the new data to the existing DataFrame
        df_analyser = pd.concat([df, new_data], ignore_index=True)

    # Return the updated DataFrame
    return df_analyser
```

Execution of the algorithm

```
# The probability used in the epsilon-greedy algorithm to determine
# whether to explore (choose a random arm) or exploit (choose the best arm
so far).
epsilon_values = [0.01]
learning_params = [0.01]

# Prepare empty lists to store results
epsilon_greedy_cumulative_regret, ucb_cumulative_regret,
exp3_cumulative_regret = [], [], []

# Prepare empty lists to store results
epsilon_greedy_results, ucb_results, exp3_results = [], [], []


# Run simulations for each combination of epsilon and learning parameters
for epsilon_val in epsilon_values:
    for learning_param in learning_params:
        epsilon_greedy_cumulative_regret =
epsilon_greedy_policy(epsilon_val, num_arms, num_iterations,
success_params, failure_params)
        ucb_cumulative_regret = UCB_algorithm(num_arms, num_iterations,
success_params, failure_params)
        exp3_cumulative_regret = EXP3_algorithm(num_arms, num_iterations,
success_params, failure_params, learning_param)
        plot_cumulative_regret(epsilon_greedy_cumulative_regret,
ucb_cumulative_regret, exp3_cumulative_regret, epsilon_val,
learning_param)
        epsilon_greedy_results.append((epsilon_val, learning_param,
epsilon_greedy_cumulative_regret))
```

ID6002W: Online Learning and Reinforcement Learning
Mid Term Exam
Course Instructor: Arun Rajkumar.
Exam Date: 18-06-2023
Prepared By: Aloy Banerjee
Roll Number: CH22M503

```
        ucb_results.append((epsilon_val, learning_param,
ucb_cumulative_regret))
        exp3_results.append((epsilon_val, learning_param,
exp3_cumulative_regret))
```



Loading the data in dataframe for analysis

```
# Create an empty DataFrame
df_Analyser = pd.DataFrame(columns=['Algorithm', 'Epsilon',
'Learning_Rate', 'Max_Regret',
                                    'Max_Regret_Iter', 'Min_Regret',
'Min_Regret_Iter'])

# Process results for each algorithm and add to the DataFrame
df_Analyser = process_results(df_Analyser, epsilon_greedy_results,
'Epsilon-Greedy', epsilon_values[0],
                              learning_params[0])
df_Analyser = process_results(df_Analyser, ucb_results, 'UCB',
epsilon_values[0], learning_params[0])
df_Analyser = process_results(df_Analyser, exp3_results, 'Exp 3',
epsilon_values[0], learning_params[0])
```

ID6002W: Online Learning and Reinforcement Learning
Mid Term Exam
Course Instructor: Arun Rajkumar.
Exam Date: 18-06-2023
Prepared By: Aloy Banerjee
Roll Number: CH22M503

```
df_Analyser_sorted = df_Analyser.sort_values(['Epsilon', 'Learning_Rate'],
ascending=[True, True])
```

Viewed the sorted dataframe containing the information about the regret for different algorithm for selected Epsilon and Learning rate

```
df_Analyser_sorted

        Algorithm  Epsilon  Learning_Rate  Max_Regret Max_Regret_Iter  \
0  Epsilon-Greedy     0.01           0.01    2.842063             993
1             UCB     0.01           0.01  161.713203             990
2           Exp 3     0.01           0.01   76.372691             895

   Min_Regret Min_Regret_Iter
0         0.0               1
1         0.0               1
2         0.3               1
```

Part 2: Comment on your observations about the regret plots obtained in the previous part. (2.5 points)

Conclusion of Part 1:

- Looking at the plot for three algorithm reveals that the value of cumulative regret over the number of rounds,
  - for **UCB**: Remain in increasing trend.
  - for **Exp-3**: Initially the regret value increased but eventually it flattened out after a certain round.
  - for **Epsilon Greedy**: On the contrary cumulative regret remains very close to zero and changes very slowly.
- We are getting the same indication from the table displayed in the last cell also.

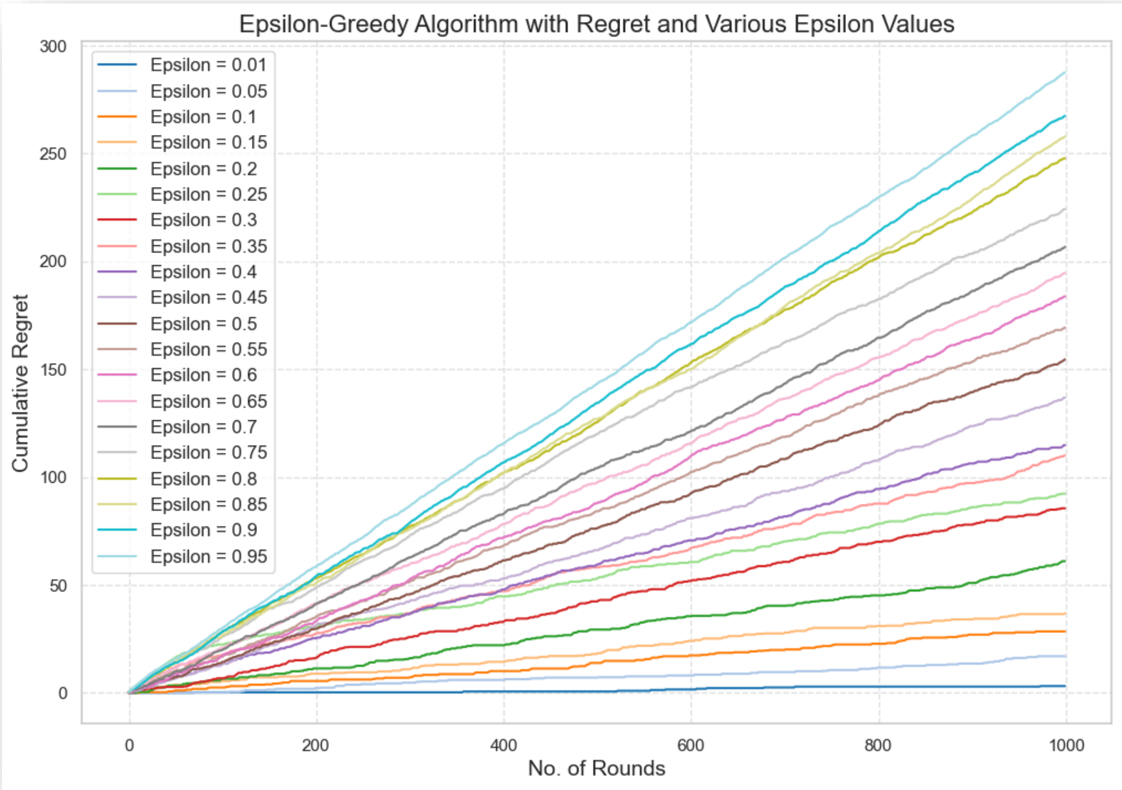
Part 3: Execution of the algorithm for different values of ϵ

```
# Different epsilon values to explore
epsilon_values_to_explore = [0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35,
0.4, 0.45, 0.5,
                             0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9,
0.95]

# List to store the cumulative regrets for each epsilon value
cumulative_regrets_for_each_epsilon = []
```

ID6002W: Online Learning and Reinforcement Learning
Mid Term Exam
Course Instructor: Arun Rajkumar.
Exam Date: 18-06-2023
Prepared By: Aloy Banerjee
Roll Number: CH22M503

```
# Run the epsilon-greedy policy for each epsilon value and store the
cumulative regrets
for epsilon in epsilon_values_to_explore:
    cumulative_regret = epsilon_greedy_policy(epsilon, num_arms,
num_iterations, success_params, failure_params)
    cumulative_regrets_for_each_epsilon.append(cumulative_regret)

# Set the style and context of the plot to make it look more attractive
sns.set(style='whitegrid', context='notebook')

# Set the color palette for the plot
colors = [mat.colormaps['tab20'](iloop) for iloop in range(20)]

# Create a larger figure
plt.figure(figsize=(12, 8))

# Loop over epsilon values
for i, epsilon in enumerate(epsilon_values_to_explore):
    # Plot the cumulative regret for each epsilon, with a different color
and line style for each one
    plt.plot(cumulative_regrets_for_each_epsilon[i], color=colors[i],
linestyle='-', label=f"Epsilon = {epsilon}")

# Set labels and title with larger fonts
plt.xlabel("No. of Rounds", fontsize=14)
plt.ylabel("Cumulative Regret", fontsize=14)
plt.title("Epsilon-Greedy Algorithm with Regret and Various Epsilon
Values", fontsize=16)

# Display the legend and increase its font size
plt.legend(fontsize=12)

# Add a slight grid for better readability
plt.grid(True, linestyle='--', alpha=0.6)

# Show the plot
plt.show()
```

ID6002W: Online Learning and Reinforcement Learning
Mid Term Exam
Course Instructor: Arun Rajkumar.
Exam Date: 18-06-2023
Prepared By: Aloy Banerjee
Roll Number: CH22M503



Conclusion Part 3:

- Various epsilon values (**0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95**) demonstrate various regret patterns in the Epsilon-Greedy algorithm. In terms of minimizing regrets and observing that the regret is growing its slope as and when we are raising the epsilon value over the number of rounds, smaller epsilon values (for example, 0.01) tend to perform better than bigger epsilon values (for example, 0.95). This suggests that a more experimental strategy (lower epsilon) can eventually result in greater overall performance.

End of Report