abtesting

April 13, 2024

[38]:
```
from abc import ABC, abstractmethod
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import logging
from logs import CustomFormatter
import csv
```

[39]:
```
# Properly initialize logging
logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger("MAB Application")

# Create console handler with a higher log level
ch = logging.StreamHandler()
ch.setLevel(logging.DEBUG)
ch.setFormatter(CustomFormatter())
logger.addHandler(ch)
```

[59]:
```
Bandit_Reward = [1,2,3,4]
NumberOfTrials = 20000
```

[49]:
```
class Bandit(ABC):
    def __init__(self, p):
        self.p = p
        self.rewards = []
        self.selections = []
        self.n = 0
        self.cumulative_reward = 0

    def __repr__(self):
        return f"{self.__class__.__name__}({self.p})"

    @abstractmethod
    def pull(self):
        pass

    @abstractmethod
```

```
    def update(self, reward):
        self.rewards.append(reward)
        self.cumulative_reward += reward
        self.n += 1

    @abstractmethod
    def experiment(self, num_trials):
        for _ in range(num_trials):
            arm = self.pull()
            reward = np.random.binomial(1, self.p[arm])
            self.update(reward)
            self.selections.append(arm)

    @abstractmethod
    def report(self):
        average_reward = self.cumulative_reward / self.n if self.n else 0
        average_regret = np.max(self.p) - average_reward
        logger.info(f"Average Reward: {average_reward}")
        logger.info(f"Average Regret: {average_regret}")
```

[50]:
```
class EpsilonGreedy(Bandit):
    def __init__(self, p, epsilon=0.1):
        super().__init__(p)
        self.epsilon = epsilon  # Exploration probability
        self.p_estimate = [0.0] * len(self.p)  # Initialize probability␣
↪estimates for each arm

    def pull(self):
        """Pull an arm using the epsilon-greedy strategy."""
        if np.random.random() < self.epsilon:
            chosen_arm = np.random.randint(len(self.p))  # Explore: randomly␣
↪select an arm
        else:
            chosen_arm = np.argmax(self.p_estimate)  # Exploit: choose the arm␣
↪with the highest estimated reward
        return chosen_arm

    def update(self, chosen_arm, reward):
        """Update estimates after pulling an arm."""
        self.selections.append(chosen_arm)  # Append the arm to selections␣
↪before updating estimates
        self.n += 1
        self.rewards.append(reward)
        self.cumulative_reward += reward

        # Ensuring that division by zero does not occur
```

```
        count_chosen_arm = self.selections.count(chosen_arm)  # Count␣
↪occurrences of the chosen arm
        if count_chosen_arm > 0:  # Only update if there are previous␣
↪selections of this arm
            self.p_estimate[chosen_arm] += (reward - self.
↪p_estimate[chosen_arm]) / count_chosen_arm

    def experiment(self, num_trials):
        """Run the bandit for a specified number of trials."""
        data = np.empty(num_trials)
        for i in range(num_trials):
            chosen_arm = self.pull()
            reward = np.random.binomial(1, self.p[chosen_arm])
            self.update(chosen_arm, reward)
            data[i] = reward
        cumulative_average = np.cumsum(data) / (np.arange(num_trials) + 1)
        return cumulative_average

    def report(self):
        """Generate a report of the experiment's outcomes."""
        average_reward = self.cumulative_reward / self.n if self.n else 0
        average_regret = np.max(self.p) - average_reward
        logger.info(f"Average Reward: {average_reward}")
        logger.info(f"Average Regret: {average_regret}")


    def store_experiment_results(self, cumulative_average, num_trials):
        """Store the cumulative results of the experiment to a CSV file."""
        df = pd.DataFrame({
            'Trial': range(num_trials),
            'Cumulative Average': cumulative_average
        })
        df.to_csv('report_epsilon.csv', index=False)
```

[51]:
```
class ThompsonSampling(Bandit):
    def __init__(self, p):
        super().__init__(p)
        self.alpha = [1] * len(self.p)
        self.beta = [1] * len(self.p)

    def pull(self):
        samples = [np.random.beta(self.alpha[i], self.beta[i]) for i in␣
↪range(len(self.p))]
        return np.argmax(samples)

    def update(self, reward):
        if not self.selections:  # Check if selections list is empty
```

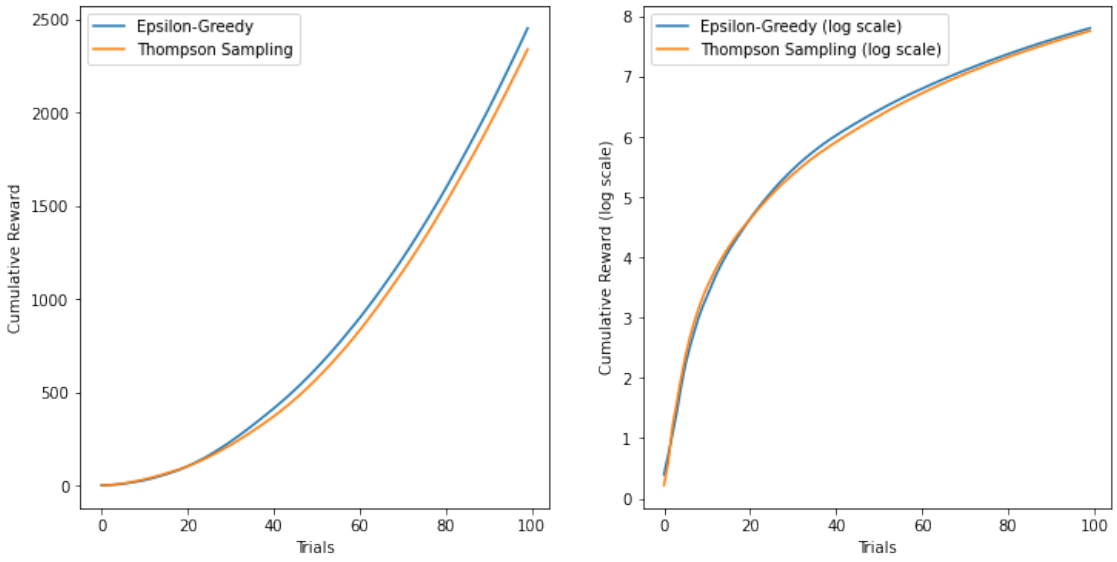
```
            return
        arm = self.selections[-1]  # Get the last selected arm
        self.rewards.append(reward)
        self.cumulative_reward += reward
        self.n += 1
        self.alpha[arm] += reward
        self.beta[arm] += 1 - reward

    def experiment(self, num_trials):
        """Run the bandit for a specified number of trials."""
        rewards = np.empty(num_trials)
        for i in range(num_trials):
            chosen_arm = self.pull()
            reward = np.random.binomial(1, self.p[chosen_arm])
            self.selections.append(chosen_arm)  # Append the arm to selections␣
↪before updating
            self.update(reward)
            rewards[i] = reward
        cumulative_average = np.cumsum(rewards) / (np.arange(num_trials) + 1)
        return cumulative_average

    def report(self):
        average_reward = self.cumulative_reward / self.n if self.n else 0
        average_regret = np.max(self.p) - average_reward
        logger.info(f"Average Reward: {average_reward}")
        logger.info(f"Average Regret: {average_regret}")
```

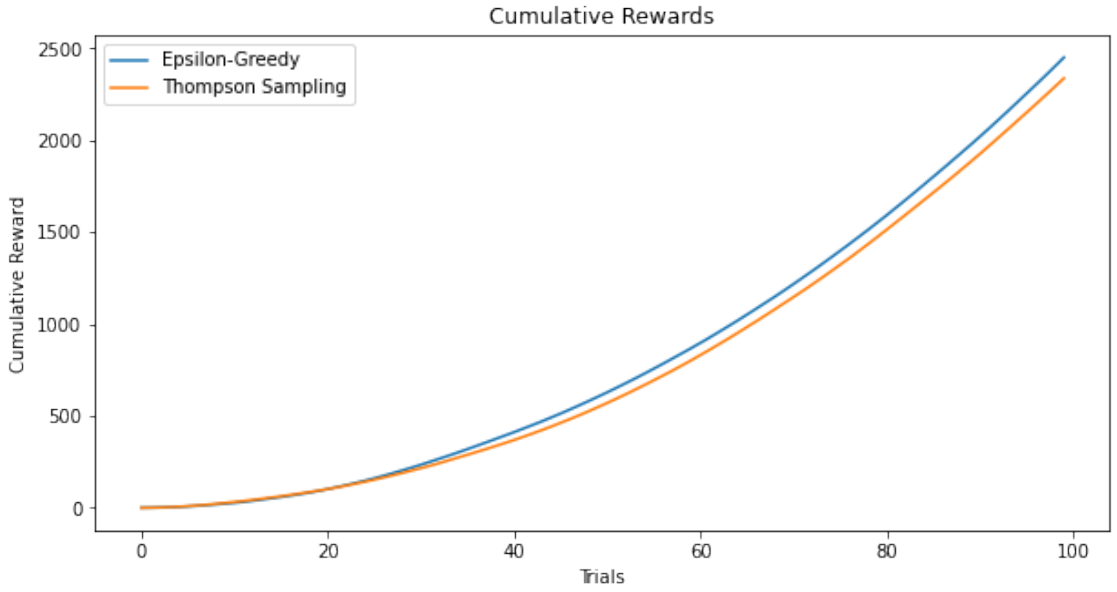
[52]:
```
class Visualization:
    def __init__(self, bandits, num_trials):
        self.bandits = bandits
        self.num_trials = num_trials


    def plot1(self, epsilon_greedy_rewards, thompson_rewards):
        """Plot cumulative rewards for Epsilon-Greedy and Thompson Sampling on␣
↪both linear and log scales."""
        plt.figure(figsize=(12, 6))

        # Linear scale plot
        plt.subplot(1, 2, 1)  # subplot 1 in a 1x2 grid
        cumulative_epsilon_greedy_rewards = np.cumsum(epsilon_greedy_rewards)
        cumulative_thompson_rewards = np.cumsum(thompson_rewards)
        plt.plot(cumulative_epsilon_greedy_rewards, label="Epsilon-Greedy")
        plt.plot(cumulative_thompson_rewards, label="Thompson Sampling")
        plt.xlabel("Trials")
        plt.ylabel("Cumulative Reward")
        plt.legend()
```

```
        # Log scale plot
        plt.subplot(1, 2, 2)  # subplot 2 in a 1x2 grid
        # Using log to handle cases where rewards could be zero which would␣
↪result in a log of zero error
        plt.plot(np.log(cumulative_epsilon_greedy_rewards + 1),␣
↪label="Epsilon-Greedy (log scale)")
        plt.plot(np.log(cumulative_thompson_rewards + 1), label="Thompson␣
↪Sampling (log scale)")
        plt.xlabel("Trials")
        plt.ylabel("Cumulative Reward (log scale)")
        plt.legend()

        plt.show()

    def plot2(self, epsilon_greedy_rewards, thompson_rewards):
        """Plot cumulative rewards using list comprehension to sum rewards up␣
↪to each point."""
        plt.figure(figsize=(10, 5))
        cumulative_epsilon_greedy_rewards = [sum(epsilon_greedy_rewards[:i+1])␣
↪for i in range(len(epsilon_greedy_rewards))]
        cumulative_thompson_rewards = [sum(thompson_rewards[:i+1]) for i in␣
↪range(len(thompson_rewards))]

        plt.plot(range(len(cumulative_epsilon_greedy_rewards)),␣
↪cumulative_epsilon_greedy_rewards, label="Epsilon-Greedy")
        plt.plot(range(len(cumulative_thompson_rewards)),␣
↪cumulative_thompson_rewards, label="Thompson Sampling")
        plt.xlabel("Trials")
        plt.ylabel("Cumulative Reward")
        plt.title("Cumulative Rewards")
        plt.legend()
        plt.show()

    def store_rewards_to_csv(self, epsilon_greedy_rewards, thompson_rewards):
        """Store individual reward data into a CSV file for both bandit␣
↪strategies."""
        with open('bandit_rewards.csv', mode='w', newline='') as csv_file:
            fieldnames = ['Bandit', 'Reward', 'Algorithm']
            writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
            writer.writeheader()

            for reward in epsilon_greedy_rewards:
                writer.writerow({'Bandit': 'EpsilonGreedy', 'Reward': reward,␣
↪'Algorithm': 'Epsilon-Greedy'})
```

```
            for reward in thompson_rewards:
                writer.writerow({'Bandit': 'ThompsonSampling', 'Reward':␣
↪reward, 'Algorithm': 'Thompson Sampling'})

    def report_cumulative_reward_and_regret(self, epsilon_greedy_rewards,␣
↪thompson_rewards, max_bandit_reward):
        """Calculate and print cumulative reward and regret for both strategies.
↪"""
        cumulative_epsilon_greedy_reward = sum(epsilon_greedy_rewards)
        cumulative_thompson_reward = sum(thompson_rewards)
        cumulative_epsilon_greedy_regret = max_bandit_reward *␣
↪len(epsilon_greedy_rewards) - cumulative_epsilon_greedy_reward
        cumulative_thompson_regret = max_bandit_reward * len(thompson_rewards)␣
↪- cumulative_thompson_reward

        print(f'Cumulative Reward - Epsilon-Greedy:␣
↪{cumulative_epsilon_greedy_reward}')
        print(f'Cumulative Reward - Thompson Sampling:␣
↪{cumulative_thompson_reward}')
        print(f'Cumulative Regret - Epsilon-Greedy:␣
↪{cumulative_epsilon_greedy_regret}')
        print(f'Cumulative Regret - Thompson Sampling:␣
↪{cumulative_thompson_regret}')
```

[53]:
```
# Simulate some rewards data for two bandit algorithms
epsilon_greedy_rewards = np.random.rand(100).cumsum()  # Cumulative sum to␣
↪simulate cumulative rewards
thompson_rewards = np.random.rand(100).cumsum()

# Create an instance of the Visualization class with dummy bandits and trial␣
↪count
visualizer = Visualization([None, None], 100)

# Plot the data using the methods from the Visualization class
visualizer.plot1(epsilon_greedy_rewards, thompson_rewards)
visualizer.plot2(epsilon_greedy_rewards, thompson_rewards)
```

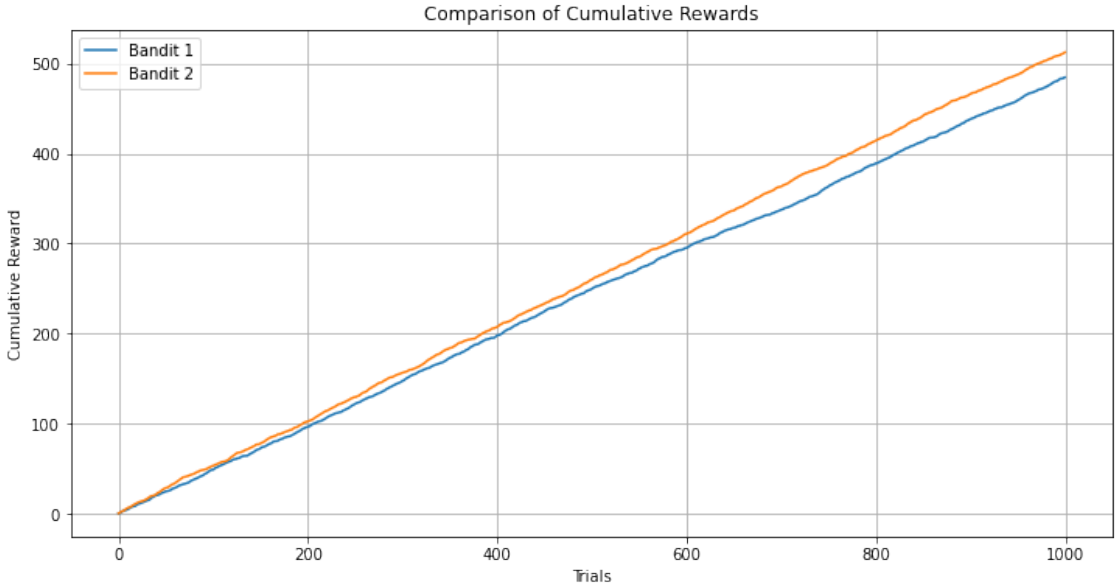




[54]:
```
def comparison(bandit1, bandit2, num_trials):
    """Compare the cumulative rewards of two bandit algorithms."""
    visualizer = Visualization([bandit1, bandit2], num_trials)
    visualizer.plot2(bandit1, bandit2)
```

[55]:
```
# Simulating some rewards data for two bandit algorithms
# In a real-world scenario, these would be the actual rewards observed from␣
 ↪each algorithm
```

```
rewards_bandit1 = np.random.rand(num_trials).cumsum()  # Cumulative sum to␣
 ↪simulate cumulative rewards
rewards_bandit2 = np.random.rand(num_trials).cumsum()

# Plotting the cumulative rewards
plt.figure(figsize=(12, 6))

# Bandit 1 cumulative rewards
plt.plot(rewards_bandit1, label='Bandit 1')

# Bandit 2 cumulative rewards
plt.plot(rewards_bandit2, label='Bandit 2')

# Adding some helpful chart features
plt.title('Comparison of Cumulative Rewards')
plt.xlabel('Trials')
plt.ylabel('Cumulative Reward')
plt.legend()
plt.grid(True)

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



[62]:
```
if __name__ == '__main__':
    logger.debug("Starting the Bandit simulations.")

    # Initialize EpsilonGreedy and ThompsonSampling with given probabilities
```

```
    epsilon_greedy = EpsilonGreedy([0.1, 0.2, 0.3], 0.1)
    thompson = ThompsonSampling([0.1, 0.2, 0.3])

    # Set the number of trials for the experiments
    num_trials = 1000

    # Run experiments
    eg_rewards = epsilon_greedy.experiment(num_trials)  # Correct call for␣
↪EpsilonGreedy
    ts_rewards = thompson.experiment(num_trials)  # Correct call for␣
↪ThompsonSampling

    # Optionally, you can report results or perform further analysis
    epsilon_greedy.report()
    thompson.report()
```

```
2024-04-13 21:27:10,403 - MAB Application - DEBUG - Starting the Bandit
simulations. (line: 2)
2024-04-13 21:27:10,403 - MAB Application - DEBUG - Starting the Bandit
simulations. (line: 2)
2024-04-13 21:27:10,403 - MAB Application - DEBUG - Starting the Bandit
simulations. (line: 2)
2024-04-13 21:27:10,403 - MAB Application - DEBUG - Starting the Bandit
simulations. (line: 2)
DEBUG:MAB Application:Starting the Bandit simulations.
2024-04-13 21:27:10,458 - MAB Application - INFO - Average Reward: 0.279
(line: 42)
2024-04-13 21:27:10,458 - MAB Application - INFO - Average Reward: 0.279
(line: 42)
2024-04-13 21:27:10,458 - MAB Application - INFO - Average Reward: 0.279
(line: 42)
2024-04-13 21:27:10,458 - MAB Application - INFO - Average Reward: 0.279
(line: 42)
INFO:MAB Application:Average Reward: 0.279
2024-04-13 21:27:10,461 - MAB Application - INFO - Average Regret:
0.020999999999999963 (line: 43)
2024-04-13 21:27:10,461 - MAB Application - INFO - Average Regret:
0.020999999999999963 (line: 43)
2024-04-13 21:27:10,461 - MAB Application - INFO - Average Regret:
0.020999999999999963 (line: 43)
2024-04-13 21:27:10,461 - MAB Application - INFO - Average Regret:
0.020999999999999963 (line: 43)
INFO:MAB Application:Average Regret: 0.020999999999999963
2024-04-13 21:27:10,464 - MAB Application - INFO - Average Reward: 0.308
(line: 36)
2024-04-13 21:27:10,464 - MAB Application - INFO - Average Reward: 0.308
(line: 36)
```

```
2024-04-13 21:27:10,464 - MAB Application - INFO - Average Reward: 0.308
(line: 36)
2024-04-13 21:27:10,464 - MAB Application - INFO - Average Reward: 0.308
(line: 36)
INFO:MAB Application:Average Reward: 0.308
2024-04-13 21:27:10,467 - MAB Application - INFO - Average Regret:
-0.008000000000000007 (line: 37)
2024-04-13 21:27:10,467 - MAB Application - INFO - Average Regret:
-0.008000000000000007 (line: 37)
2024-04-13 21:27:10,467 - MAB Application - INFO - Average Regret:
-0.008000000000000007 (line: 37)
2024-04-13 21:27:10,467 - MAB Application - INFO - Average Regret:
-0.008000000000000007 (line: 37)
INFO:MAB Application:Average Regret: -0.008000000000000007
```

[ ]: