

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 class Bandit:
5     def __init__(self, k):
6         self.k = k
7         self.means = np.random.normal(0, np.sqrt(3), k)
8
9     def pull(self, arm):
10         return np.random.normal(self.means[arm], 1)
11
12 def epsilon_greedy(bandit, epsilon, steps):
13     k = bandit.k
14     Q = np.zeros(k)
15     N = np.zeros(k)
16     rewards = np.zeros(steps)
17
18     for t in range(steps):
19         if np.random.rand() < epsilon:
20             arm = np.random.choice(k)
21         else:
22             arm = np.argmax(Q)
23
24         reward = bandit.pull(arm)
25         N[arm] += 1
26         Q[arm] += (reward - Q[arm]) / N[arm]
27         rewards[t] = reward
28
29     return rewards
30
31 def greedy_optimistic(bandit, Q1, steps):
32     k = bandit.k
33     Q = np.ones(k) * Q1
34     N = np.zeros(k)
35     rewards = np.zeros(steps)
36
37     for t in range(steps):
38         arm = np.argmax(Q)
39         reward = bandit.pull(arm)
40         N[arm] += 1
41         Q[arm] += (reward - Q[arm]) / N[arm]
42         rewards[t] = reward
43
44     return rewards
45
46 def ucb(bandit, c, steps):
47     k = bandit.k
48     Q = np.zeros(k)
49     N = np.zeros(k)
50     rewards = np.zeros(steps)
51
52     for t in range(steps):
53         if t < k:
```

```

54         arm = t
55     else:
56         ucb_values = Q + c * np.sqrt(np.log(t + 1) / (N + 1e-5))
57         arm = np.argmax(ucb_values)
58
59     reward = bandit.pull(arm)
60     N[arm] += 1
61     Q[arm] += (reward - Q[arm]) / N[arm]
62     rewards[t] = reward
63
64     return rewards
65
66 def run_simulation(algorithm, bandit, param, steps, runs):
67     all_rewards = np.zeros((runs, steps))
68     for run in range(runs):
69         rewards = algorithm(bandit, param, steps)
70         all_rewards[run] = rewards
71     return np.mean(all_rewards, axis=0)
72
73 def main():
74     # Parameters
75     k = 10
76     steps = 1000
77     runs = 100
78
79     # Initialize bandit
80     bandit = Bandit(k)
81
82     # Run simulations
83     epsilon_rewards = run_simulation(epsilon_greedy, bandit, 0.1, steps,
runs)
84     optimistic_rewards = run_simulation(greedy_optimistic, bandit, 5, steps,
runs)
85     ucb_rewards = run_simulation(ucb, bandit, 2, steps, runs)
86
87     # Plot results
88     plt.figure(figsize=(10, 6))
89     plt.plot(epsilon_rewards, label='Epsilon-Greedy ($\epsilon=0.1$)',
color='lightseagreen')
90     plt.plot(optimistic_rewards, label='Optimistic Greedy (Q1=5)',
color='purple')
91     plt.plot(ucb_rewards, label='UCB (c=2)', color='deeppink')
92     plt.xlabel('Steps')
93     plt.ylabel('Average Reward')
94     plt.legend()
95     plt.title('Average Reward over Time')
96     plt.savefig('Average_Reward_over_Time.png')
97     plt.show()
98
99
100     # Part 2: Summary of Comparison Plot with different hyperparameters
101
102     # Hyperparameter values
103     epsilon_values = [0.01, 0.1, 0.2, 0.3]
104     Q1_values = [5, 3, 1, 0.5]

```

```
105     c_values = [0.1, 0.5, 1, 2]
106
107     # Average rewards for different hyperparameters
108     epsilon_rewards = [np.mean(run_simulation(epsilon_greedy, bandit,
109 epsilon, steps, runs)) for epsilon in epsilon_values]
110     Q1_rewards = [np.mean(run_simulation(greedy_optimistic, bandit, Q1,
111 steps, runs)) for Q1 in Q1_values]
112     c_rewards = [np.mean(run_simulation(ucb, bandit, c, steps, runs)) for c
113 in c_values]
114
115     # Plot results
116     plt.figure(figsize=(10, 6))
117     plt.plot(epsilon_values, epsilon_rewards, label='$\epsilon$-greedy',
118 color='lightseagreen')
119     plt.plot(Q1_values, Q1_rewards, label='Optimistic Greedy ',
120 color='purple')
121     plt.plot(c_values, c_rewards, label='UCB', color='deeppink')
122     plt.xscale('log', base=2)
123     plt.xlabel('$\epsilon \backslash \text{quad} / \backslash \text{quad} c \backslash \text{quad} / \backslash \text{quad} Q_0$')
124     plt.ylabel('Average reward over first 1000 steps')
125     plt.legend()
126     plt.title('Summary comparison of algorithms')
127     plt.savefig('Summary_comparison_of_algorithms.png')
128     plt.show()
129
130 if __name__ == '__main__':
131     main()
```