Assignment -1

10-Armed Bandit Testbed

```
class TenArmedBandit:
    def __init__(self, k=10):
        self.k = k
        self.q_star = np.random.normal(0, 1, k)
        self.best_action = np.argmax(self.q_star)

    def step(self, action):
        reward = np.random.normal(self.q_star[action], 1)
        return reward
```

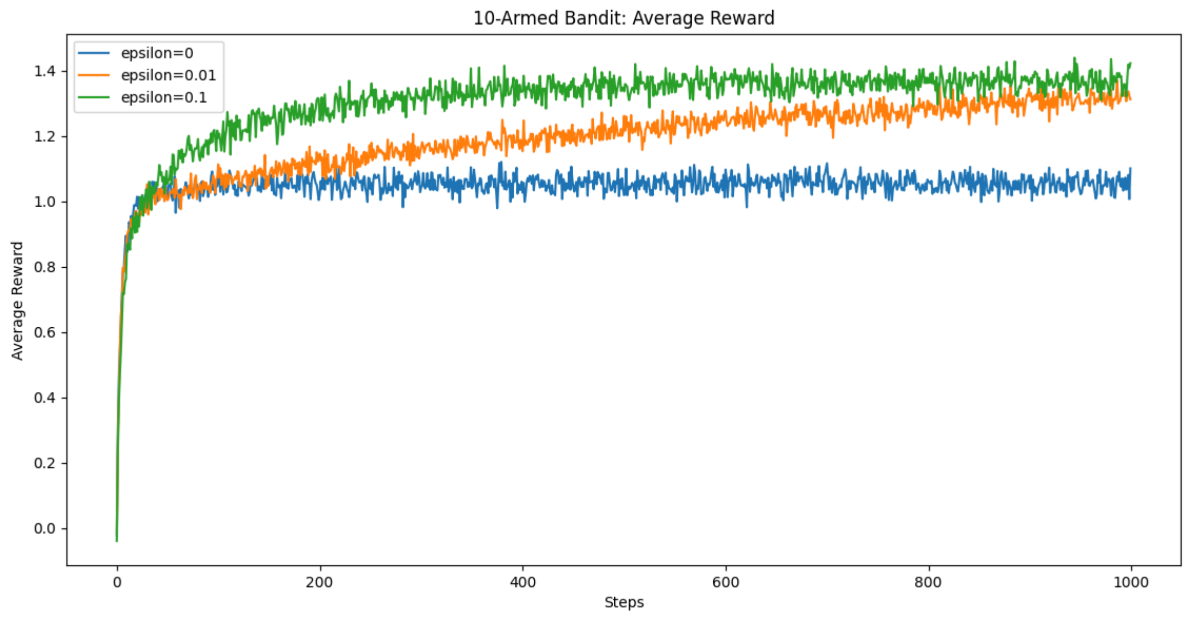
Greedy and ϵ-greedy Algorithms

```
# Agent

class Agent:
    def __init__(self, n_actions=10, epsilon=0):
        self.n_actions = n_actions
        self.epsilon = epsilon
        self.Q = np.zeros(n_actions)
        self.N = np.zeros(n_actions)

    def select_action(self):
        if np.random.rand() < self.epsilon:
            return np.random.randint(self.n_actions)
        return np.argmax(self.Q)

    def update(self, action, reward):
        self.N[action] += 1
        self.Q[action] += (reward - self.Q[action]) / self.N[action]
```

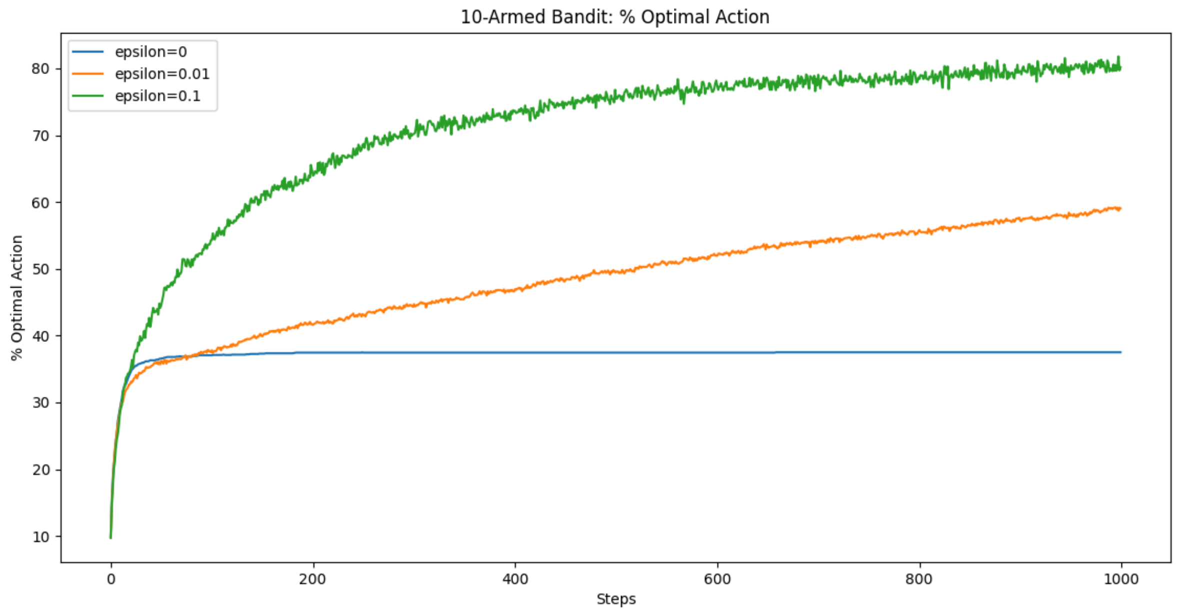
```
# Simulation Function
def simulate(bandits, agent_class, steps=1000, epsilon=0, c=np.sqrt(2)):
    avg_rewards = np.zeros(steps)
    optimal_action_counts = np.zeros(steps)

    for bandit in bandits:
        n_actions = len(bandit.q_star)
        agent = agent_class(n_actions=n_actions, epsilon=epsilon) if
agent_class==Agent else agent_class(n_actions=n_actions, c=c)

        for t in range(steps):
            action = agent.select_action()
            reward = bandit.step(action)
            agent.update(action, reward)

            avg_rewards[t] += reward
            if action == bandit.best_action:
                optimal_action_counts[t] += 1

    avg_rewards /= len(bandits)
    optimal_action_counts = (optimal_action_counts / len(bandits)) * 100
    return avg_rewards, optimal_action_counts


# number of bandits & steps
n_bandits = 2000
steps = 1000

# Generate bandits
bandits_10 = [TenArmedBandit() for _ in range(n_bandits)]

# Greedy and epsilon-greedy
epsilons = [0, 0.01, 0.1]
results = {}
```

```
for eps in epsilons:
    avg_r, opt_a = simulate(bandits_10, Agent, steps=steps, epsilon=eps)
    results[f'eps_{eps}'] = (avg_r, opt_a)
    print(f"Completed epsilon={eps}")
```

Average reward vs. time



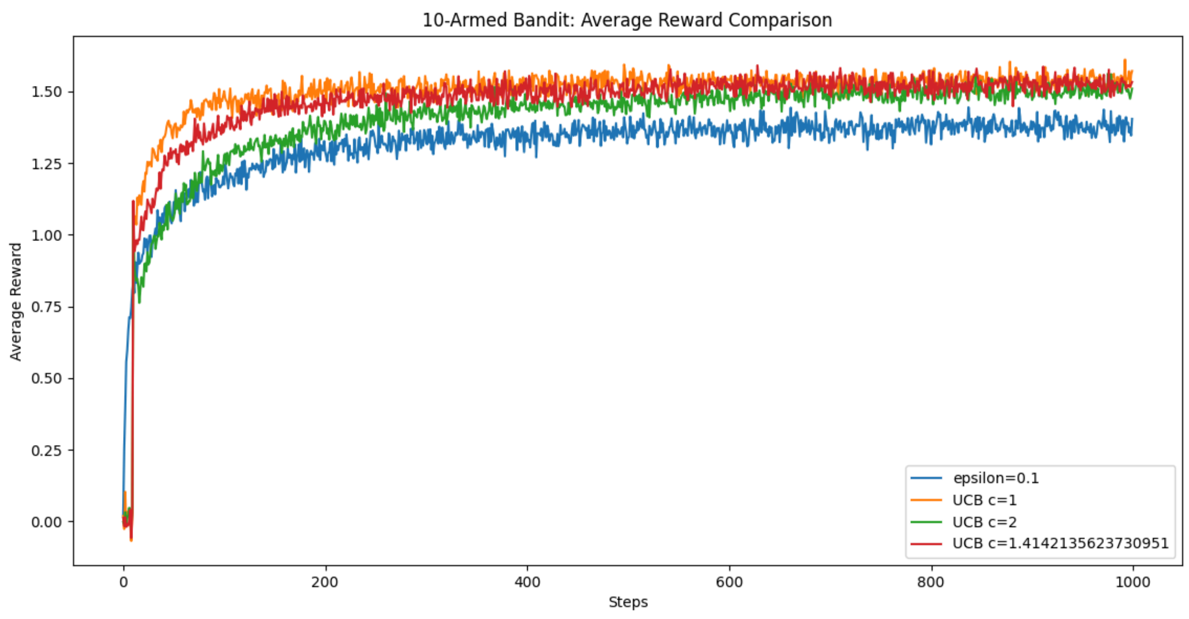
% Optimal action vs. time



Upper Confidence Bound (UCB) Algorithm

```
# UCBAgent



class UCBAgent:
    def __init__(self, n_actions=10, c=np.sqrt(2)):
        self.n_actions = n_actions
        self.c = c
        self.Q = np.zeros(n_actions)
        self.N = np.zeros(n_actions)
        self.time = 0

    def select_action(self):
        self.time += 1
        ucb_values = self.Q + self.c * np.sqrt(np.log(self.time + 1) /
(self.N + 1e-5))
        return np.argmax(ucb_values)

    def update(self, action, reward):
        self.N[action] += 1
        self.Q[action] += (reward - self.Q[action]) / self.N[action]

# different c's
c_values = [1, 2, np.sqrt(2)]
ucb_results = {}

for c in c_values:
    avg_r, _ = simulate(bandits_10, UCBAgent, steps=steps, c=c)
    ucb_results[f'c_{c}'] = avg_r
    print(f"Completed UCB with c={c}")
```

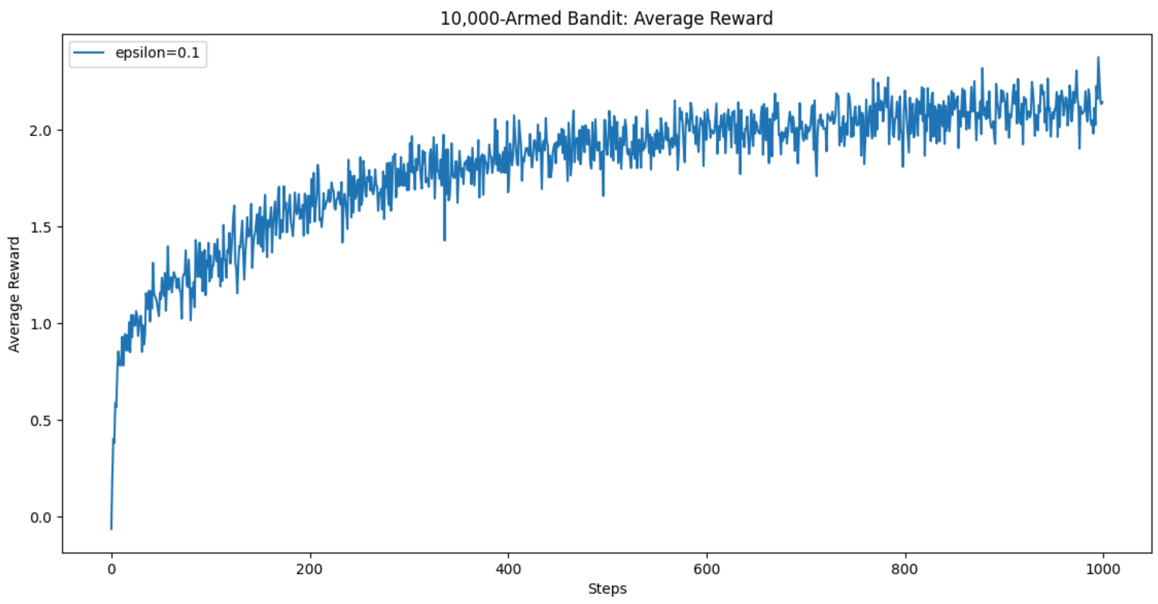
Average performance of UCB action selection on the 10-armed testbed



Large Bandit Testbed

```
# Bandit Environment
class LargeBandit:
    def __init__(self, k=10000):
        self.k = k
        self.q_star = np.random.normal(0, 1, k)
        self.best_action = np.argmax(self.q_star)

    def step(self, action):
        reward = np.random.normal(self.q_star[action], 1)
        return reward

# Large-Scale 10,000-Armed Bandit
n_bandits_large = 200
bandits_large = [LargeBandit() for _ in range(n_bandits_large)]

# Greedy and epsilon=0.1 for large bandit
avg_r_large, opt_a_large = simulate(bandits_large, Agent, steps=steps,
epsilon=0.1)
```

10,000-Armed Bandit



Inferences and Conjectures

Algorithm	Behavior	Insights
Greedy (ε=0)	Exploits the first good action found; often suboptimal.	Poor long-term reward due to lack of exploration.
ε-Greedy (ε=0.01)	Mostly exploits, occasionally explores.	Performs well if initial estimates are close to true values.
ε-Greedy (ε=0.1)	Balances exploration and exploitation better.	Achieves higher average reward over time.
UCB (c≈√2)	Dynamically explores uncertain actions.	Often outperforms ε-greedy; efficient exploration.
Large Bandit (10,000 arms)	Sparse optimal actions.	ε=0.1 helps discover good arms; exploration is crucial.