# 0_Simple_approach

August 4, 2025

## 1 Simple aproach to Multi-armed bandit problem

Here I would like to compare three basic approaches to the Multi-armed bandit problem - **Epsilon-Greedy**, **Upper Confidence Bound**, and **Thompson Sampling**. Their implementation codes were taken from this source: https://www.geeksforgeeks.org/machine-learning/multi-armed-bandit-problem-in-reinforcement-learning/.

### 1.1 Aproaches

#### 1.1.1 Epsilon-Greedy

[1]:
```
class EpsilonGreedy:
    def __init__(self, n_arms, epsilon):
        self.n_arms = n_arms
        self.epsilon = epsilon
        self.counts = np.zeros(n_arms)  # Number of times each arm is pulled
        self.values = np.zeros(n_arms)  # Estimated values of each arm

    def select_arm(self):
        if np.random.rand() < self.epsilon:
            return np.random.randint(0, self.n_arms)
        else:
            return np.argmax(self.values)

    def update(self, chosen_arm, reward):
        self.counts[chosen_arm] += 1
        n = self.counts[chosen_arm]
        value = self.values[chosen_arm]
        self.values[chosen_arm] = ((n - 1) / n) * value + (1 / n) * reward
```

#### 1.1.2 Upper Confidence Bound

[2]:
```
class UCB:
    def __init__(self, n_arms):
        self.n_arms = n_arms
        self.counts = np.zeros(n_arms)
        self.values = np.zeros(n_arms)
        self.total_counts = 0
    def select_arm(self):
        ucb_values = self.values + np.sqrt(2 * np.log(self.total_counts + 1) /
↪(self.counts + 1e-5))
        return np.argmax(ucb_values)

    def update(self, chosen_arm, reward):
        self.counts[chosen_arm] += 1
        self.total_counts += 1
        n = self.counts[chosen_arm]
        value = self.values[chosen_arm]
        self.values[chosen_arm] = ((n - 1) / n) * value + (1 / n) * reward
```

#### 1.1.3 Thompson Sampling

[3]:
```
class ThompsonSampling:
    def __init__(self, n_arms):
        self.n_arms = n_arms
        self.successes = np.zeros(n_arms)
        self.failures = np.zeros(n_arms)

    def select_arm(self):
        sampled_values = np.random.beta(self.successes + 1, self.failures + 1)
        return np.argmax(sampled_values)

    def update(self, chosen_arm, reward):
        if reward > 0:
            self.successes[chosen_arm] += 1
        else:
            self.failures[chosen_arm] += 1
```

### 1.2 Simulation

In this simulation the goal is to compare basic properties of theese there aproaches.

Let's define parameters of the simulation: - `n_arms`: number of arms - `n_simulations`: Number of simulation repetitions - `n_steps`: Number of steps in one run - `epsilon`: Parameter for Epsilon-Greedy aproach

[4]:
```
# Libraties
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import seaborn as sns
```

[5]:
```
# Function for simulation
```

```python
def simulate_deterministic(agent_class, agent_kwargs, true_rewards, n_steps, n_simulations):
    n_arms = len(true_rewards)
    rng_sim = np.random.default_rng()
    total_rewards = np.zeros((n_simulations, n_steps))
    records = []

    algorithm_label = f"{agent_class.__name__} {agent_kwargs}"
    for sim in range(n_simulations):
        agent = agent_class(**agent_kwargs)
        reward_cum = 0

        for t in range(n_steps):
            # Agent
            arm = agent.select_arm()
            reward = true_rewards[arm] # no noise, same reward all the time
            agent.update(arm, reward)

            # Save rewards
            reward_cum += reward
            total_rewards[sim, t] = reward

            # Save record
            records.append({
                "Simulation": sim,
                "Step": t,
                "Algorithm": algorithm_label,
                **agent_kwargs,
                "Reward": reward,
                "Cumulative Reward": reward_cum
            })

    # Calculate some statistics
    cumulative_rewards = np.cumsum(total_rewards, axis=1) # Kumulativní odměny v čase pro každou simulaci
    mean_rewards = np.mean(cumulative_rewards, axis=0) # Průměrný průběh kumulativní odměny napříč simulacemi
    std_rewards = np.std(cumulative_rewards[:, -1]) # Směrodatná odchylka celkové odměny na konci (stabilita výkonu)
    final_mean = np.mean(cumulative_rewards[:, -1]) # Průměrná celková odměna na konci simulací (výkon algoritmu)

    return {
    "mean_rewards": mean_rewards,
    "std_rewards": std_rewards,
    "final_mean": final_mean,
    "records_df": pd.DataFrame(records)
}
```

[6]:
```python
# Parameters
n_arms = 10
true_rewards = np.array([0.2, 0.5, 0.8, 0.1, 0.3, 0.4, 0.7, 0.6, 0.25, 0.05])
n_steps = 1000
n_simulations = 1000
```

[7]:
```python
# Run the simulation
results = {}
```

[8]:
```python
results["Epsilon-Greedy ( =0.1)"] = simulate_deterministic(
    EpsilonGreedy, {"n_arms": n_arms, "epsilon": 0.1}, true_rewards, n_steps,␣
 ↪n_simulations
)
```

[9]:
```python
results["UCB"] = simulate_deterministic(
    UCB, {"n_arms": n_arms}, true_rewards, n_steps, n_simulations
)
```

[10]:
```python
results["Thompson Sampling"] = simulate_deterministic(
    ThompsonSampling, {"n_arms": n_arms}, true_rewards, n_steps, n_simulations
)
```

[11]:
```python
# Check the results
df_all = pd.concat([result["records_df"] for result in results.values()],␣
 ↪ignore_index=True)
# df_all.head()
df_all
```

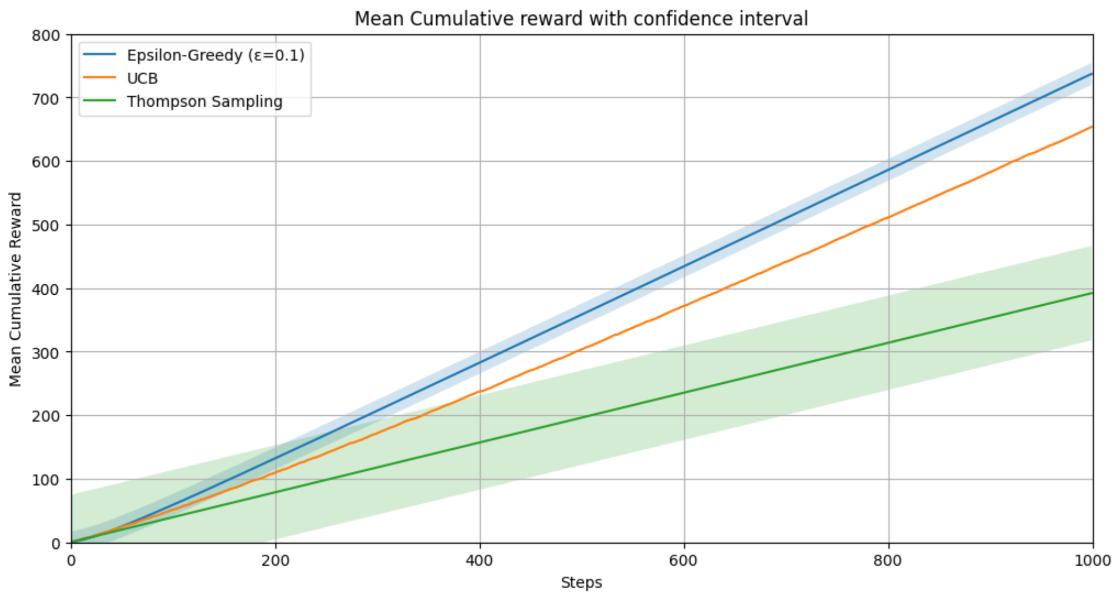
[11]:
```
         Simulation  Step                                  Algorithm  \
0                 0     0  EpsilonGreedy {'n_arms': 10, 'epsilon': 0.1}
1                 0     1  EpsilonGreedy {'n_arms': 10, 'epsilon': 0.1}
2                 0     2  EpsilonGreedy {'n_arms': 10, 'epsilon': 0.1}
3                 0     3  EpsilonGreedy {'n_arms': 10, 'epsilon': 0.1}
4                 0     4  EpsilonGreedy {'n_arms': 10, 'epsilon': 0.1}
…               …   …                                      …
2999995         999   995             ThompsonSampling {'n_arms': 10}
2999996         999   996             ThompsonSampling {'n_arms': 10}
2999997         999   997             ThompsonSampling {'n_arms': 10}
2999998         999   998             ThompsonSampling {'n_arms': 10}
2999999         999   999             ThompsonSampling {'n_arms': 10}

         n_arms  epsilon  Reward  Cumulative Reward
0            10      0.1    0.20               0.20
1            10      0.1    0.20               0.40
2            10      0.1    0.20               0.60
```

```
3             10      0.1    0.20              0.80
4             10      0.1    0.20              1.00
…          …       …      …                …
2999995       10      NaN    0.20            359.95
2999996       10      NaN    0.20            360.15
2999997       10      NaN    0.40            360.55
2999998       10      NaN    0.25            360.80
2999999       10      NaN    0.10            360.90

[3000000 rows x 7 columns]
```

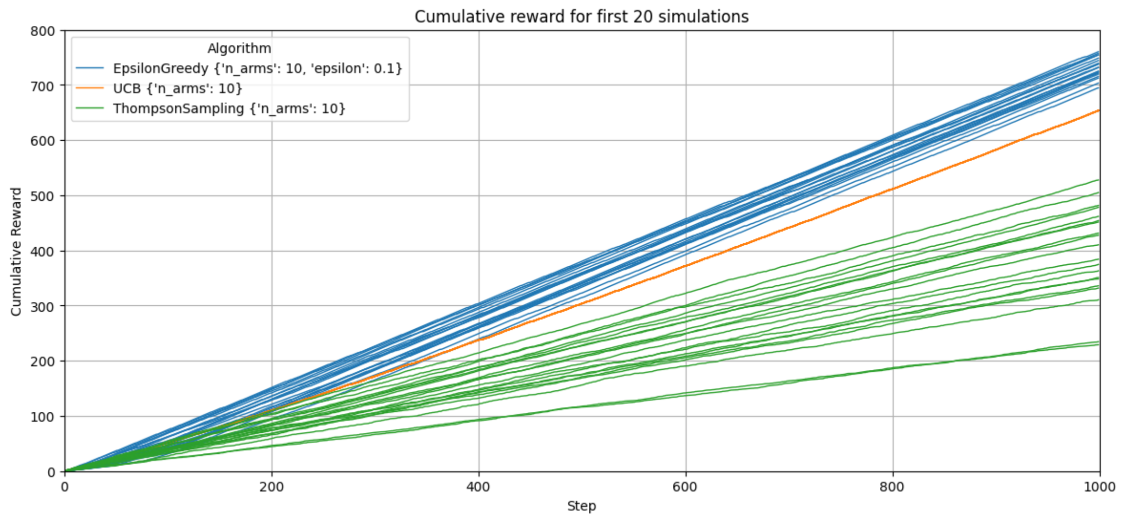
[12]:
```
# Print the mean and std for the aproaches
for name, result in results.items():
    print(f"{name}: Final mean = {result['final_mean']:.2f}, Std =␣
 ↪{result['std_rewards']:.2f}")
```

```
Epsilon-Greedy ( =0.1): Final mean = 737.06, Std = 17.25
UCB: Final mean = 653.60, Std = 0.00
Thompson Sampling: Final mean = 392.07, Std = 74.25
```

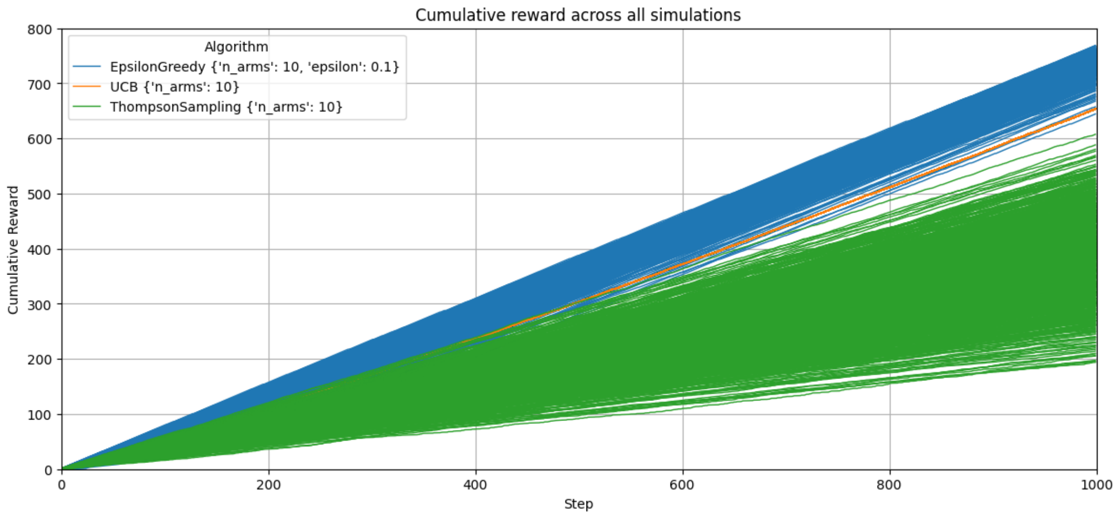
[13]:
```
plt.figure(figsize=(12, 6))

for label, result in results.items():
    mean_rewards = result["mean_rewards"]
    std_rewards = result["std_rewards"]

    plt.plot(mean_rewards, label=label)
    plt.fill_between(range(n_steps),
                     mean_rewards - std_rewards,
                     mean_rewards + std_rewards,
                     alpha=0.2)
plt.xlabel("Steps")
plt.ylabel("Mean Cumulative Reward")
plt.title("Mean Cumulative reward with confidence interval")
plt.legend()
plt.xlim(0, 1000)
plt.ylim(0, 800)
plt.ylim(bottom=0)
plt.grid(True)
# plt.show()
```



[14]:
```
# Visualize first 20 simulations
df_plot = df_all[df_all["Simulation"] < 20]

plt.figure(figsize=(14, 6))
sns.lineplot(
    data=df_plot,
    x="Step",
    y="Cumulative Reward",
    hue="Algorithm",
    units="Simulation",
    estimator=None,
    lw=1
)
plt.title("Cumulative reward for first 20 simulations")
plt.xlim(0, 1000)
plt.ylim(0, 800)
plt.ylim(bottom=0)
plt.grid(True)
# plt.show()
```



[15]:
```python
# Visualize all simulations
df_plot = df_all

plt.figure(figsize=(14, 6))
sns.lineplot(
    data=df_plot,
    x="Step",
    y="Cumulative Reward",
    hue="Algorithm",
    units="Simulation",
    estimator=None,
    lw=1
)
plt.title("Cumulative reward across all simulations")
plt.xlim(0, 1000)
plt.ylim(0, 800)
plt.ylim(bottom=0)
plt.grid(True)
# plt.show()
```


Cumulative reward across all simulations
Algorithm
EpsilonGreedy {'n_arms': 10, 'epsilon': 0.1}
UCB {'n_arms': 10}
ThompsonSampling {'n_arms': 10}
Cumulative Reward
Step
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: