SI140A Probability and Statistics Final Project: Performance Evaluation of Bandit Learning Algorithms

Jingran Fan Anrui Wang Zhao Lu

Due date: 2025/1/12 10:59pm

Abstract

Multi-armed bandit problems are fundamental models in sequential decision-making under uncertainty, wherein an agent must choose from several options (arms) over repeated trials to maximize cumulative rewards. These problems capture the delicate balance between exploration—seeking information about less-known options—and exploitation—leveraging current knowledge to select the best option. Classical bandit learning algorithms, such as ϵ-greedy, Upper Confidence Bound (UCB), and Thompson Sampling (TS), have been extensively studied and form the cornerstone of modern reinforcement learning techniques. More recently, Bayesian approaches that incorporate prior beliefs and update them with observed data have gained traction, offering theoretical elegance and robust performance in a variety of settings.

This project focuses on evaluating the performance of well-known bandit algorithms through numerical experiments. In Part I, we consider classical bandit algorithms operating on Bernoulli arms with parameters provided by an oracle. Although the oracle's parameters and optimal attainable reward (the "oracle value") are unknown to the algorithms, they provide a ground truth reference for performance comparison. We implement and benchmark ϵ-greedy (with various ϵ-values), UCB (with different confidence scales), and TS (with varying Beta priors) under identical experimental conditions. By analyzing their regret—defined as the gap between the algorithm's cumulative reward and the oracle value—we investigate how tuning parameters and prior knowledge impacts the exploration-exploitation trade-off.

In the optional Part II, we extend our analysis to a Bayesian bandit setting with discounted rewards, where prior distributions on arm parameters are continuously updated as more data is observed. We examine intuitive heuristics and discuss why these heuristics may fail to achieve optimality. Furthermore, we explore the derivation of optimal policies through recursive equations and investigate practical methods for exact and approximate solutions.

Overall, this project aims to provide a rigorous empirical evaluation of both classical and Bayesian bandit algorithms. By comparing their performance and understanding their underlying trade-offs, we gain deeper insights into bandit learning theory and develop intuition for selecting and designing effective strategies in diverse applications.

Contents

Introduction

In many real-world scenarios—from online advertising to medical trials—decision-makers must choose actions to maximize cumulative rewards. However, these decisions also provide valuable information that can guide future actions. This creates a fundamental tension between exploiting what we already know to gain immediate benefits and exploring new options to potentially improve future outcomes. This challenge is central to the field of reinforcement learning and is known as the exploration-exploitation trade-off.

A classic illustration is the multi-armed bandit problem. Imagine walking into a casino and facing a slot machine with multiple arms, each offering a different, unknown payoff distribution. Your goal is to pull the arms over a sequence of trials to earn as many rewards as possible. Since you do not know which arm is best, you must try them out (exploration) while continuing to play the arm that seems most promising (exploitation). Crucially, the reward probabilities remain fixed but hidden, and the only way to learn them is by experimenting.

This report examines three classical strategies for solving the multi-armed bandit problem—ϵ-greedy, Upper Confidence Bound (UCB), and Thompson Sampling (TS). By comparing their performance, we gain insight into how they balance exploration and exploitation and how well they adapt to uncertain, reward-driven environments.

Part I: Classical Bandit Algorithms

Problem 1

Choose $N = 5000$ and compute the theoretically maximized expectation of aggregate rewards over N time slots. Suppose we have an oracle that provides the parameters of the Bernoulli distributions for three arms as follows:

$$\theta_1 = 0.7, \quad \theta_2 = 0.5, \quad \theta_3 = 0.4.$$

We choose the time horizon as $N = 5000$ time steps. If we know these parameters beforehand (as the oracle does), the strategy to maximize the expected total reward is to always pull the arm with the highest success probability, which in this case is arm 1 (with $\theta_1 = 0.7$).

The expected reward is:

$$\max_{I(t), t=1,\ldots,N} \mathbb{E}\left[\sum_{t=1}^{N} r_{I(t)}\right] = N \times \theta_1 = 5000 \times 0.7 = 3500.$$

Thus, the theoretically maximized expectation is: $\boxed{3500.}$

Problem 2

Imports and parameters:

```
import matplotlib.pyplot as plt
import numpy as np
import random, math, copy

# Set random seed for reproducibility
np.random.seed(42)

num_arms = 3
theta = np.array([0.7, 0.5, 0.4])
```

Implementation of ϵ-greedy algorithm:

```
def epsilon_greedy(epsilon, N, theta):
    Q = np.zeros(num_arms)
    counts = np.zeros(num_arms)

    rewards_history = []

    for t in range(1, N+1):
        # Choose arm using epsilon-greedy
        if np.random.rand() < epsilon:
            # Exploration: choose a random arm
            arm = np.random.randint(num_arms)
        else:
            # Exploitation: choose the best arm so far
            arm = np.argmax(Q)

        # Simulate pulling the chosen arm and get reward
        reward = 1 if np.random.rand() < theta[arm] else 0

        # Update counts and estimates
        counts[arm] += 1
        Q[arm] = Q[arm] + (1/counts[arm])*(reward - Q[arm])

        rewards_history.append(reward)

    return rewards_history
```

Implementation of UCB algorithm:

```
def ucb(c, N, theta):
    num_arms = len(theta)
    # Estimated values for each arm
    Q = np.zeros(num_arms)
    # Count of how many times each arm is pulled
    counts = np.zeros(num_arms)

    rewards_history = []

    # Initialization: pull each arm once
    for t in range(num_arms):
        arm = t
        reward = 1 if np.random.rand() < theta[arm] else 0
        counts[arm] += 1
        Q[arm] = reward
        rewards_history.append(reward)

    # Main loop
    for t in range(num_arms+1, N+1):
        # Compute UCB values
        ucb_values = Q + c * np.sqrt((2*np.log(t)) / counts)

        arm = np.argmax(ucb_values)

        reward = 1 if np.random.rand() < theta[arm] else 0
        counts[arm] += 1
        Q[arm] += (1/counts[arm])*(reward - Q[arm])
        rewards_history.append(reward)

    return np.array(rewards_history)
```

Implementation of Thompson Sampling algorithm:

```
from scipy.stats import beta

# Two sets of prior parameters for TS
# Set 1: (1,1), (1,1), (1,1)
alpha_set_1 = np.array([1, 1, 1])
beta_set_1 = np.array([1, 1, 1])

# Set 2: (601,401), (401,601), (2,3)
```

```
alpha_set_2 = np.array([601, 401, 2])
beta_set_2 = np.array([401, 601, 3])

def thompson_sampling_history(N, theta, alpha_init, beta_init):
    alpha = alpha_init.copy()
    beta_ = beta_init.copy()
    rewards_history = []

    for t in range(1, N+1):
        sampled_thetas = [np.random.beta(alpha[j], beta_[j]) for j in range(num_arm
        arm = np.argmax(sampled_thetas)
        reward = 1 if np.random.rand() < theta[arm] else 0
        rewards_history.append(reward)
        alpha[arm] += reward
        beta_[arm] += (1 - reward)

    return np.array(rewards_history)
```

Problem 3

The results for the three algorithms with various parameters, averaged over 200 trials and 5000 time slots, are presented below:

ε-Greedy

- $\varepsilon = 0.1$: **3416.51**
- $\varepsilon = 0.5$: **3086.39**
- $\varepsilon = 0.9$: **2748.39**

Upper Confidence Bound (UCB)

- $c = 1$: **3411.85**
- $c = 5$: **2976.41**
- $c = 10$: **2826.26**

Thompson Sampling (TS)

- $(\alpha_1, \beta_1) = (1, 1), (\alpha_2, \beta_2) = (1, 1), (\alpha_3, \beta_3) = (1, 1)$: **3477.98**
- $(\alpha_1, \beta_1) = (601, 401), (\alpha_2, \beta_2) = (401, 601), (\alpha_3, \beta_3) = (2, 3)$: **3492.28**

Part II: Bayesian Bandit Algorithms