CS747 - Assignment 1 Report

Keshav Singhal 20D070047

September 10, 2023

1 Task 1: Sampling Algorithms

1.1 Epsilon Greedy Algorithm

1.1.1 Algorithm Description

The Epsilon-Greedy algorithm is a simple and widely used approach in multi-armed bandit problems. It balances exploration and exploitation by occasionally selecting a random arm (exploration) and otherwise choosing the arm with the highest estimated reward (exploitation). The algorithm maintains an estimate of the mean reward for each arm and updates these estimates based on observed rewards.

1.1.2 Implementation

The Epsilon-Greedy algorithm was already implemented in the code and follows the standard approach. A parameter $\epsilon = 0.1$ is set to control the exploration rate. At each time step, with probability ϵ, a random arm is chosen uniformly at random, and with probability $1-\epsilon$, the arm with the highest estimated reward is chosen.

1.1.3 Results

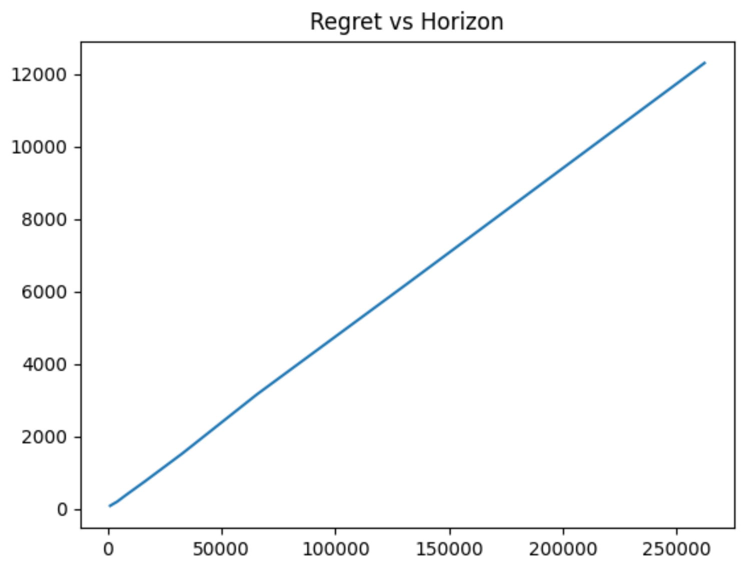

Figure 1: Cumulative Regret for Epsilon-Greedy Algorithm

The Epsilon-Greedy Algorithm is a simple algorithm and can be seen that it achieves linear regret.

1.2 Upper Confidence Bound

1.2.1 Algorithm Description

The Upper Confidence Bound (UCB) algorithm is a popular approach in multi-armed bandit problems that balances exploration and exploitation by choosing arms based on their upper confidence bounds. UCB maintains an estimate of the mean reward and a measure of uncertainty for each arm. It selects arms that have high estimated rewards and high uncertainty to encourage exploration.

1.2.2 Implementation

Our implementation of the UCB algorithm follows the standard approach. It maintains two lists for each arm: the count of pulls and the sum of rewards. To choose an arm, the algorithm calculates an upper confidence bound for each arm using the count and reward information. The arm with the highest upper confidence bound is selected for pulling.

The upper confidence bound for arm i at time t is calculated as:

$$UCB_i(t) = \hat{R}_i(t) + \sqrt{\frac{2\ln(t)}{N_i(t)}}$$

Where: $\hat{R}_i(t)$ is the estimated mean reward of arm i at time t. $N_i(t)$ is the count of pulls for arm i up to time t.

1.2.3 Results

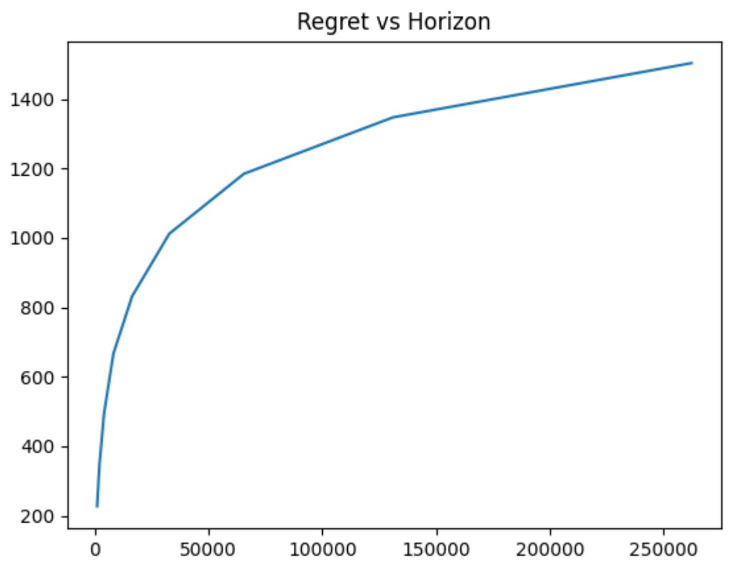

Figure 2: Cumulative Regret for UCB Algorithm

It can be seen that the UCB Algorithm achieves sub-linear regret.

1.3 Kullback Leibler - Upper Confidence Bound

1.3.1 Algorithm Description

The KL-UCB algorithm is another approach to multi-armed bandit problems that balances exploration and exploitation. It uses the Kullback-Leibler (KL) divergence to estimate the upper confidence bound for each arm. KL-UCB maintains an estimate of the mean reward and a measure of uncertainty for each arm, similar to UCB.

1.3.2 Implementation

Our implementation of the KL-UCB algorithm follows the standard approach. It maintains two lists for each arm: the count of pulls and the sum of rewards. To choose an arm, the algorithm calculates the upper confidence bound for each arm using the KL divergence-based formula. The arm with the highest upper confidence bound is selected for pulling.

The upper confidence bound for arm i at time t is calculated as:

$$KL - UCB_i(t) = \max \left\{ q \in [\hat{p}_a^t, 1] \,\middle|\, \mathrm{KL}\left(\hat{p}_a^t, q\right) \leq \frac{\ln(t) + 3\ln(\ln(t))}{N_i(t)} \right\}$$

Where: - $\hat{p}_a^t$ is the empirical mean of rewards of arm i at time t. - $N_i(t)$ is the count of pulls for arm i up to time t. - $\mathrm{KL}(p, q)$ is the Kullback-Leibler divergence between two Bernoulli distributions with probabilities p and q.

1.3.3 Results

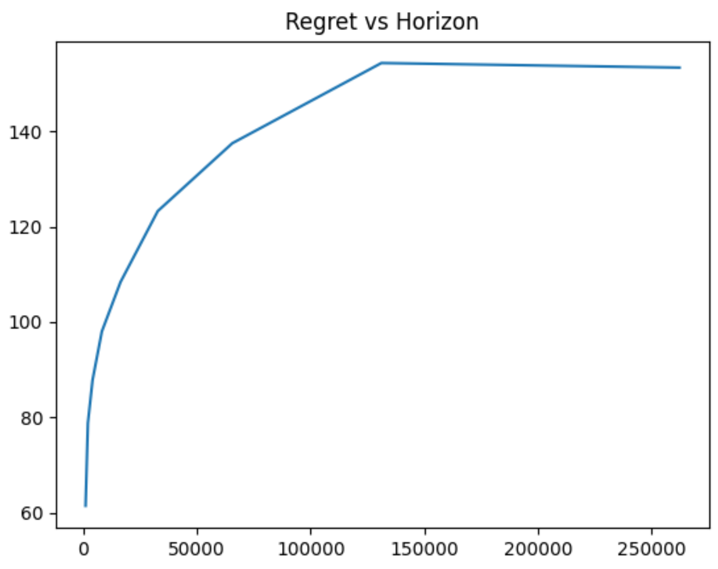

Figure 3: Cumulative Regret for KL-UCB Algorithm

Just like UCB algorithm, KL-UCB also achieves sub-linear regret but has overall lower regret over the horizon than UCB.

1.4 Thompson Sampling

1.4.1 Algorithm Description

Thompson Sampling is a popular algorithm for solving the multi-armed bandit problem. It combines probability theory with Bayesian inference to make decisions on which arms to pull. The key idea is to maintain a posterior distribution for each arm's true mean and sample from these distributions to make decisions.

1.4.2 Implementation

Our implementation of the Thompson Sampling algorithm follows the standard approach. For each arm, we maintain a Beta distribution as the posterior distribution. The algorithm samples from these Beta distributions to estimate the probability that each arm is the best. It then selects the arm with the highest estimated probability and pulls it.

At time t, consider an arm denoted as a with s_{ta} successful outcomes (1's or heads) and f_{ta} failures (0's or tails). The expression $\text{Beta}(s_{ta}+1, f_{ta}+1)$ reflects our belief about the true average of arm a. We generate a sample x_{ta} from the $\text{Beta}(s_{ta}+1, f_{ta}+1)$ distribution, where

$$\text{Beta}(\alpha, \beta) := \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$$

1.4.3 Results

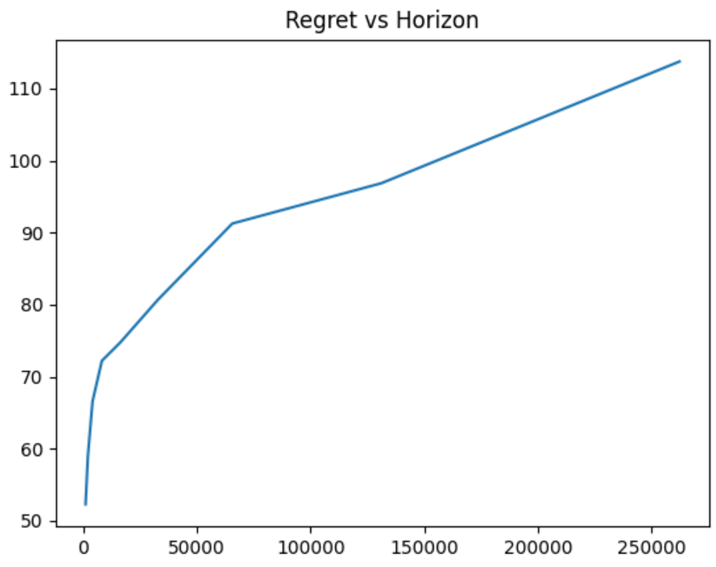

Figure 4: Cumulative Regret for Thompson Sampling Algorithm

Just like UCB and KL-UCB algorithm, Thompson Sampling also achieves sub-linear regret but has overall even lower regret over the horizon than both UCB and KL-UCB.

2 Task 2A

2.1 Problem Description

In this task, we investigate how the difference between the means of arms affects the regret accumulated by the UCB (Upper Confidence Bound) algorithm. We consider two-armed bandit instances with means represented as $[p_1, p_2]$, where the higher mean arm is fixed at p_1, and we vary the mean of the other arm, p_2, within the range of 0 to p_1. Specifically, we vary p_2 in increments of 0.05. This exploration is conducted over a horizon of 30000 time steps.

2.2 Implementation

For this assignment, we set $p_1 = 0.9$ and systematically vary p_2 from 0 to 0.9, inclusively, in steps of 0.05. At each setting of p_2, we perform a series of bandit experiments using the UCB algorithm. The regret is calculated over the entire horizon.

2.3 Results

The plot of regret as a function of p_2 reveals intriguing patterns. Initially, as p_2 increases from 0 to 0.85, the regret also increases. This trend can be attributed to the fact that the UCB algorithm is more conservative when exploring arms with lower mean values. As a result, it explores the suboptimal arm (with mean p_2) more frequently, leading to higher regret.

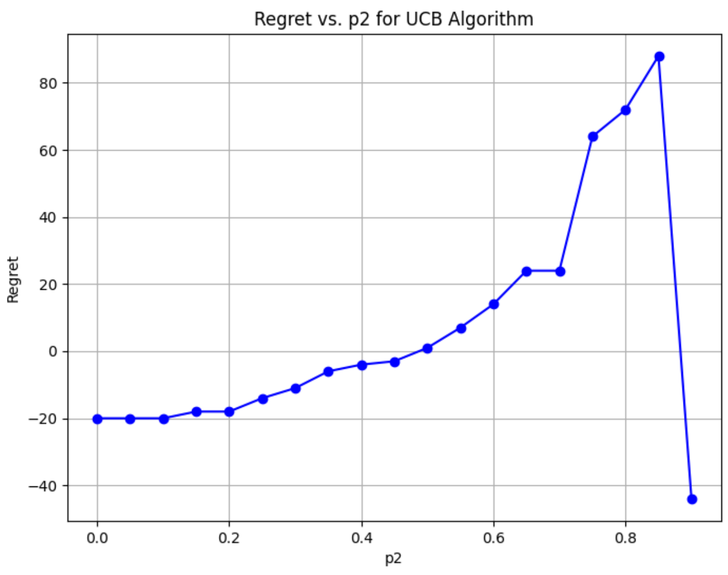

Figure 5: Regret vs P2 for UCB Algorithm

However, something interesting happens as we reach $p_2 = 0.9$. At this point, the regret immediately starts to fall. The reason behind this decline in regret is that p_2 is now equal to p_1. This means that both arms have identical mean rewards. As a result, the UCB algorithm effectively treats both arms as equally promising, leading to more balanced exploration. The negative regret for P2 = 0.9 is only due to finite run of the algorithm, expected value of regret over long horizons for P2 = 0.9 would be 0.

3 Task 2B

3.1 Problem Description

In this task, we aim to compare the behavior of the UCB (Upper Confidence Bound) and KL-UCB (Kullback-Leibler Upper Confidence Bound) algorithms on two-armed bandits. The key focus is on examining how the choice of means of the arms, while keeping the difference between the means fixed, impacts the regret accumulated by each algorithm. We maintain a fixed difference, Δ, between the means of the two arms, where $\Delta = p_1 - p_2 = 0.1$. To explore this, we vary the mean of the second arm, p_2, within the range of 0 to 0.9 (inclusive) in increments of 0.05. This experimentation is conducted over a horizon of 30000 time steps.

3.2 Results

3.2.1 UCB Algorithm

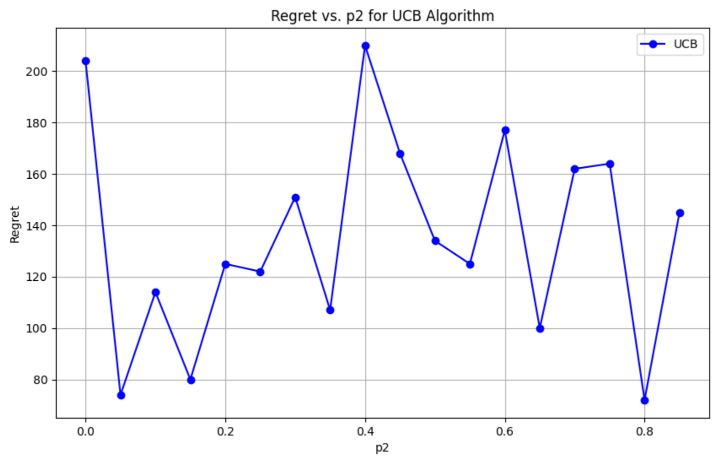

Figure 6: Regret vs P2 for UCB Algorithm

When we analyze the performance of the UCB algorithm, we observe that the regret exhibits a rather unpredictable pattern. As we vary p_2 from 0 to 0.9, the regret does not follow a consistent trend. Instead, it appears to fluctuate without a discernible pattern. This erratic behavior of regret suggests that the UCB algorithm's performance in this fixed delta setting is sensitive to the specific instances generated. The lack of a clear trend in regret indicates that UCB may struggle to adapt optimally to different scenarios, resulting in variable performance.

3.2.2 KL-UCB Algorithm

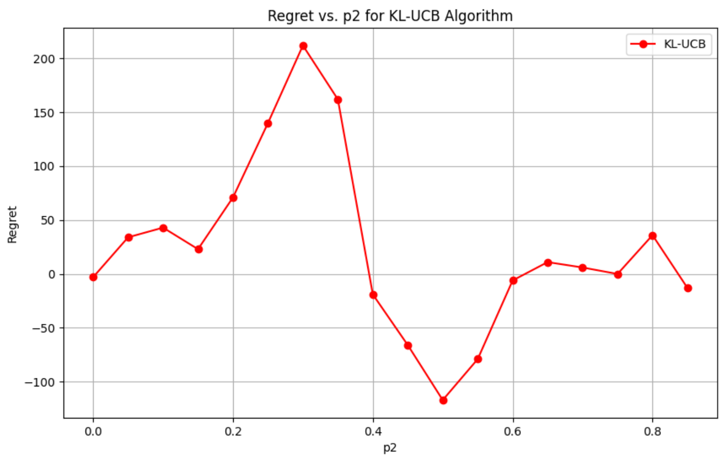

Figure 7: Regret vs P2 for KL-UCB Algorithm

The KL-UCB algorithm showcases a more structured response to changes in the means of the arms. As we vary p_2, we notice several distinct phases in the behavior of regret. Initially, from $p_2 = 0$ to around $p_2 = 0.3$, the regret consistently increases. This phase signifies that the KL-UCB algorithm is actively exploring and accumulating regret as it learns about the arm means.

However, something intriguing happens when p_2 reaches approximately 0.3. At this point, the regret begins to decline, reaching its lowest point at around $p_2 = 0.5$. The reason behind this decline is that p_2

has now become more competitive with p_1 due to their proximity. As a result, the algorithm shifts its focus towards the suboptimal arm (arm with p_2), and the regret decreases as a consequence.

However, as p_2 continues to increase beyond 0.5, the regret starts to rise once more. This suggests that, beyond a certain threshold, the algorithm's exploration of the suboptimal arm leads to higher regret.

4 Task 3: Faulty Bandit Setting

4.1 Problem Description

In Task 3, we delve into a bandit instance where our pulls are no longer guaranteed to be successful, introducing the concept of a faulty bandit setting. Here, each arm has a known probability of returning the correct output when pulled. However, there is also a possibility of the arm returning a random output of either 0 or 1 uniformly.

The primary objective of this task is to devise an effective algorithm that can maximize the accumulated reward within this environment. The algorithm must adapt to the possibility of receiving faulty outcomes while still making informed decisions to optimize reward.

4.2 Algorithm Description

To tackle the faulty bandit setting, we employ the FaultyBanditsAlgo which is a modified Thompson Sampling Algorithm including Exploration Reward and Fault Estimation.

- **Arm Selection (`give_pull`):** The algorithm selects an arm to pull based on a combination of factors. It calculates an exploration bonus to balance exploration and exploitation and samples means from a Beta distribution. The combined values from these calculations guide the arm selection process.
- **Reward Estimation (`get_reward`):** When a pull results in a reward, the algorithm updates the alpha and beta parameters for the chosen arm. If the pull is faulty (as determined by a probability), a random reward of 0 or 1 is assigned. Additionally, the algorithm maintains estimates of fault probabilities for each arm.

4.3 Implementation

The algorithm's implementation consists of maintaining hyperparameters for alpha and beta distributions, estimating fault probabilities, and incorporating exploration bonuses. It dynamically adapts to the observed rewards and faulty pulls to make arm selections.

4.4 Results

The above algorithm has been able to pass all autograded testcases with the following rewards.

- Testcase 1 PASSED. Reward: 8645.10
- Testcase 2 PASSED. Reward: 1910.54
- Testcase 3 PASSED. Reward: 3224.76

The simulator rewards are as follows: [815.18, 1676.18, 3434.16, 6927.94, 13968.06, 28043.66, 56209.34, 112473.58, 225251.88]

5 Task 4: Multi-Multi-Armed Bandit Setting

5.1 Problem Description

In Task 4, we explore the dynamic environment of a multi-multi-armed bandit setting. This unique scenario involves two bandit instances operating simultaneously. When an arm index is specified for a pull, one of the two bandit instances is chosen uniformly at random, and the corresponding arm of the selected instance is pulled. The environment then provides the obtained reward and indicates which bandit instance was chosen for that pull. Both bandit instances have an equal number of arms.

The primary challenge in this task is to design an algorithm that can effectively maximize the accumulated reward within this complex multi-multi-armed bandit setting.

5.2 Algorithm Description

To address the multi-multi-armed bandit setting, we implement the MultiBanditsAlgo algorithm which is a modified Thompson Sampling Algorithm.

- **Initialization**: The algorithm initializes variables such as the number of arms (`num_arms`) and the horizon (`horizon`). It also creates matrices to keep track of the number of successful (1) and unsuccessful (0) pulls for each arm in both bandit instances.
- **Arm Selection (`give_pull`)**: When choosing an arm to pull, the algorithm leverages the Beta distribution to estimate the likelihood of success for each arm in both instances. It samples from the Beta distribution and computes the average of these samples. The algorithm selects the arm with the highest average as the choice for the current pull.
- **Reward Handling (`get_reward`)**: Upon receiving a reward, the algorithm updates the appropriate count (1 or 0) for the chosen arm in the corresponding bandit instance based on the success or failure of the pull.

5.3 Results

The above algorithm has been able to pass all autograded testcases with the following rewards.

- Testcase 1 PASSED. Reward: 6467.98
- Testcase 2 PASSED. Reward: 1485.72
- Testcase 3 PASSED. Reward: 11001.10

The simulator rewards are as follows: [810.62, 1668.98, 3398.38, 6868.52, 13816.96, 27734.98, 55581.18, 111272.14, 222679.88]