CS 747 : Assignment 1

Sainath Vavilapalli, 200050125

September 2022

Contents

1 Epsilon Greedy Algorithm

Figure 1: Regret vs Horizon for Epsilon Greedy

I ran simulator.py using the code for epsilon greedy in the given codebase. The given epsilon greedy algorithm is of type 3, i.e, there is a predefined ϵ. For each pull, we generate a random number between 0 and 1. If the number is less than ϵ, we explore, i.e, generate a random number in the range [0, num_arms - 1] and return the index of that arm. Otherwise, we return the index of the arm with the highest empirical mean.
Performance : It gives us a linear regret, i.e, for large horizon, the ratio of regret to horizon is close to 0.05

2 Task 1

I have implemented UCB, KL-UCB and Thompson Sampling algorithms in Task 1.

2.1 UCB Algorithm

:

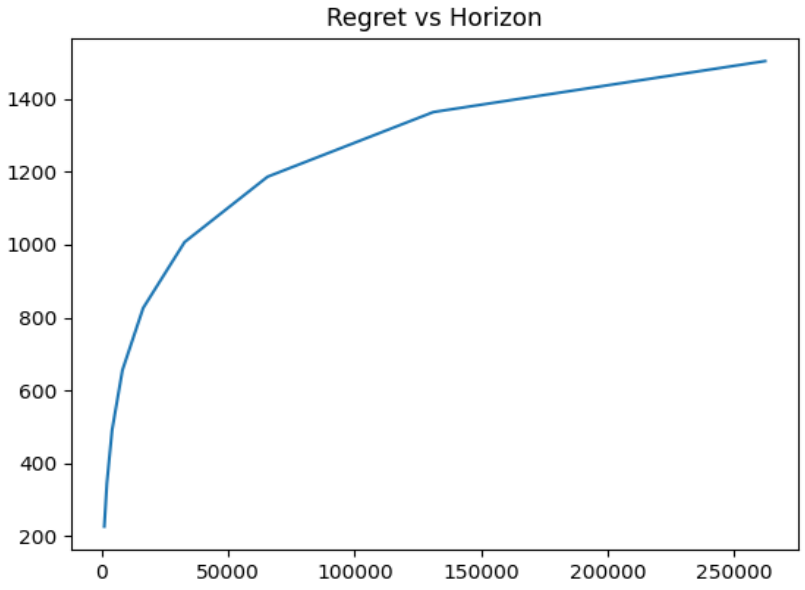

Figure 2: Regret vs Horizon for UCB

I initially implemented Round robin sampling, to ensure that every arm is pulled at least once. After that, for each pull, I calculated the ucb values for all the arms and returned the arm with the highest ucb value. The value of ucb is given by

$$\mathrm{ucb}_a^t = \hat{\mathrm{p}}_a^t + \sqrt{\frac{2\ln(\mathrm{t})}{\mathrm{u}_a^t}}$$

In the above equation, p_a^t represents the empirical probability for arm at time t and u_a^t represents the number of times an arm a has been sampled upto time t.

Performance : As we can see, ucb performs significantly better than epsilon-greedy algorithm . From the graph, it appears highly likely that the regret is sub-linear, which is indeed the case.

2.2 KL-UCB Algorithm

:

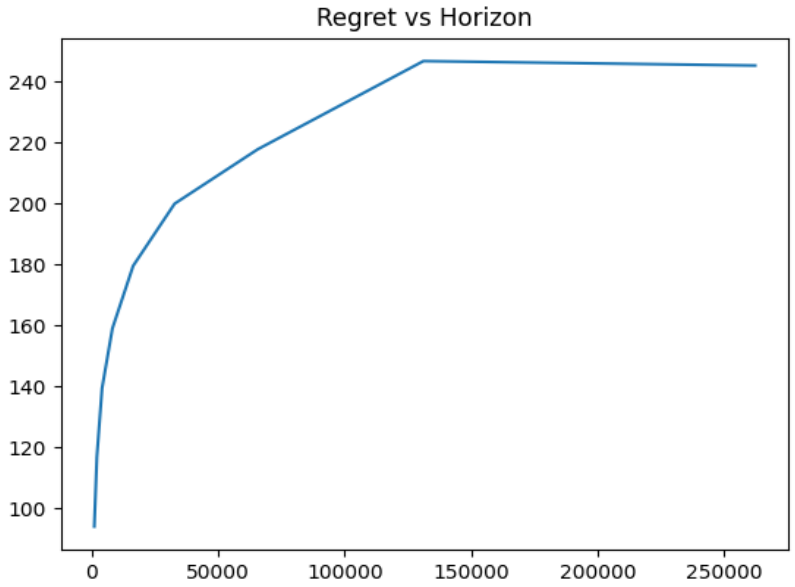

Figure 3: Regret vs Horizon for KL-UCB

Similar to UCB, I implented Round Robin sampling initially to ensure that each arm is sampled atleast once. After that, for the subsequent pulls, I calculated the kl-ucb estimate for each arm using the below equation :

$$ucb - kl_a^t = max[q \in [\hat{p}_a^t, 1] \; s.t \; u_a^t KL(\hat{p}_a^t, q) \leq ln(t) + cln(ln(t)]$$

In the above equation, c $\geq$ 3. Also, KL(x, y) denotes the KL divergence between x, y and is defined as $KL(x, y) = xln(\frac{x}{y}) + (1 - x)ln(\frac{1-x}{1-y})$.
Basically, the ucb-kl value for an arm can be found as the solution to the equation :

$$KL(p_a^t, q) = \frac{ln(t) + cln(ln(t))}{u_a^t}$$

I implemented a function solve to solve the above equation. I used Binary search to solve the equation. Initially the interval for q will be $[p_a^t, 1]$. We calculate the value of the difference in LHS and RHS of the above equation at the midpoint of the interval, and based on the sign of the difference, we update the interval accordingly. We do this until we get an interval of size 1e-6.
After obtaining the ucb-kl values, we sample the arm with the highest ucb-kl value.
Performance : As we can see, KL-UCB gives us a great improvement in performance. Not only is the regret sub-linear, but the rate of increase in regret with horizon also decreases greatly. However, there is a tradeoff. KL-UCB takes a considerably higher amount of time compared to other algorithms because of the equation solving involved.

2.3 Thompson Sampling

:

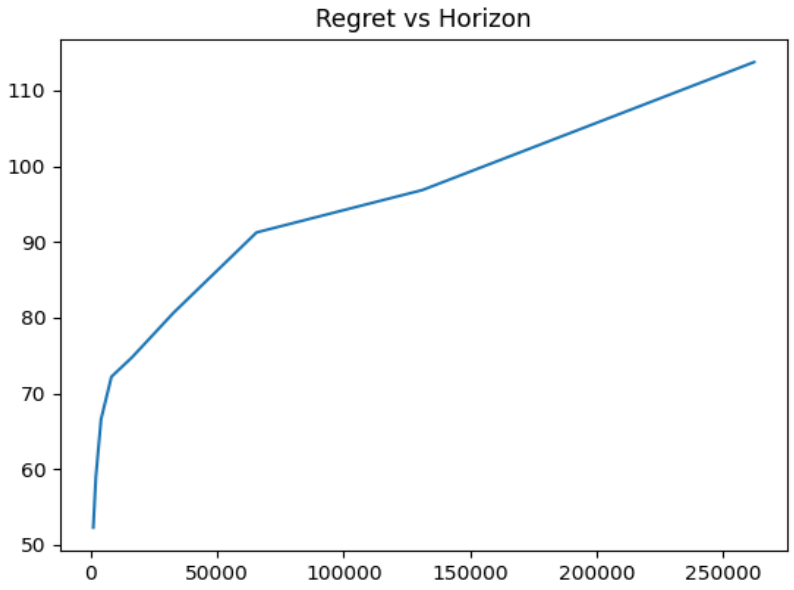

Figure 4: Regret vs Horizon for Thompson Sampling

The implementation of Thompson sampling was pretty straightforward. I maintained two arrays, successes and failures storing the number of successes and failures for each arm so far. For each pull, I obtained a sample from the following distribution for each of the arms and then returned the arm with the highest value obtained : Beta($\text{successes}_a + 1, \text{failures}_a + 1$).$\text{successes}_aand\text{failures}_a$ denote the number of successes and failures of the arm a so far.

Performance : So far, thompson sampling outperformed the other algorithms in terms of performance. Even at large values of horizon, regret incurred by Thompson sampling is far less compared to the other algorithms. Of course, the regret is sub-linear.

3 Task 2

3.1 Batched Sampling using Thompson Sampling

:

Figure 5: Regret vs Batch Size for Batched Thompson Sampling

I relied on Beta Priors and the data from previous batches for implementing this batched Thompson sampling. I maintained an array of successes and failures for all the arms in the AlgorithmBatched class object. For each batch, I used the successes and failures arrays from previous batches to sample arms. I repeated the sampling process batch$_s$ize times and maintained a dict to keep track of the arms being pulled. Then, extracted the unique arms and their respective frequencies for that concerned batch and return them.
I found a reference justifying why this method works and attached the same at the end of the pdf.

4 Task 3

4.1 Batched Sampling using Thompson Sampling

:

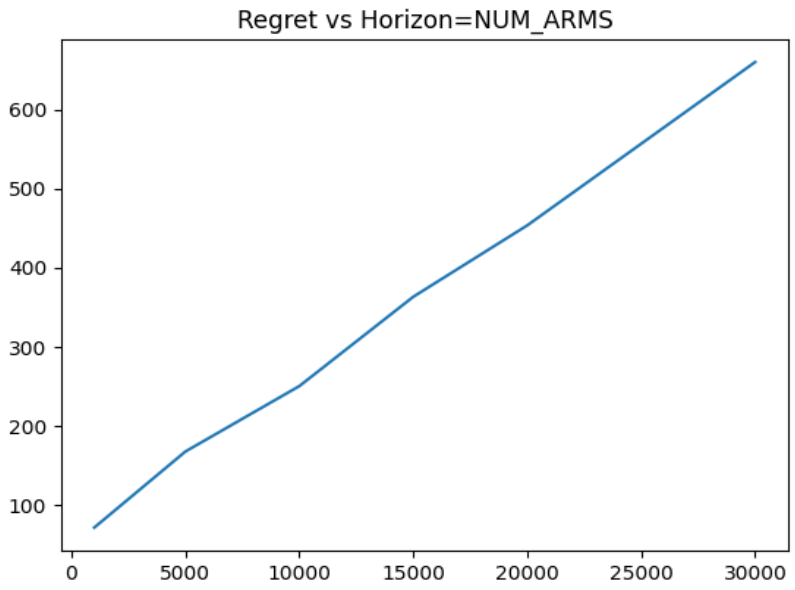

Figure 6: Regret vs Horizon for Epsilon-Greedy algo

I implemented Epsilon-Greedy algorithm of type 2 with epsilon value equal to 0.02. Value of regret is very small compared to the number of arms. However, before settling with this algorithm, I tried a variety of other algorithms, but epsilon-greedy gave superior performance compared to the others. Some other approaches I tried are :

4.2 Aternate approach 1

As we know that the distribution is uniform, I initially randomly sampled 20 arms and pulled each of them 10 times. Then, I picked the arm with maximum empirical probability among them and pulled rest of the time. I did this with the confidence that we will encounter an arm with mean with probability > 0.9, 90 percent of the time. Indeed, this is what has happened as you can see from the graph below.

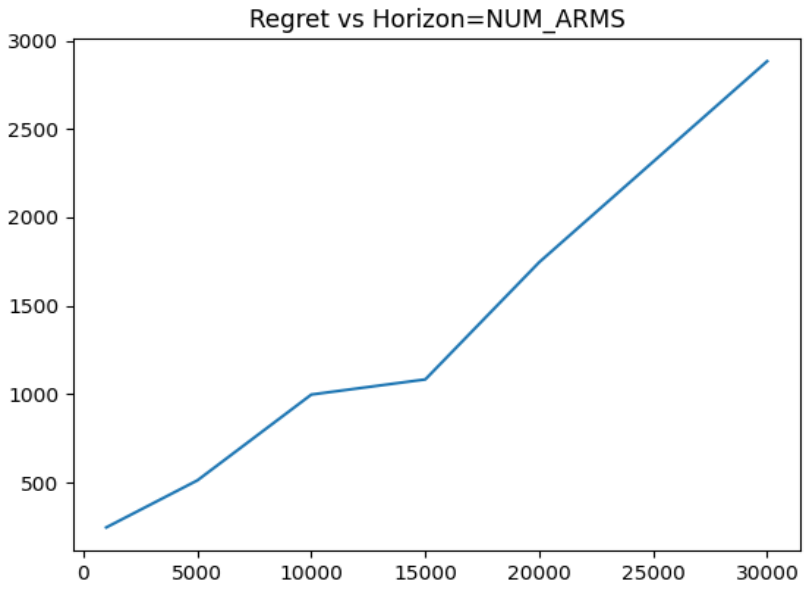

Figure 7: Regret vs Horizon for approach 1

As expected, the ratio of regret to horizon for this approach is 0.1, justifying that indeed, an arm with mean > 0.9 is being picked. However, the regret is about 5 times the regret due to Epsilon-Greedy for large values of the horizon.

4.3 Alternate approach 2 :

I directly tried Thompson sampling but I got a very large value of regret. Also, the time taken for this algorithm to run was very high, so I discarded the idea.

5 References :

- https://arxiv.org/pdf/2110.00202.pdf