CS747 Assignment-1

Rohan Gupta, Roll Number: 180010048

September 2020

Contents

1 Introduction

Each section in this report is answers the corresponding assignment task. In the beginning of each section, I briefly explain the parts of 'bandits.py', relevant to the task. I then move on to delineate the observations that could be drawn from the data generated for the task.

Re-iterating what each task comprises- Under 'Task-1', regret vs horizon plots for algorithms Epsilon greedy, UCB, KL-UCB and Thompson sampling are shown and compared in this section. Under 'Task-2', Thompson sampling and a new method Thompson sampling with hint are compared. The hint taken from instances is the value of the optimal arm's true mean. Finally, under 'Task 3', three values of epsilon are given, such that the epsilon whose value is in between other values has the least regret among them.

There were three instances containing 2, 5 and 25 arms, respectively. Hence, we have three plots for each tasks.

Note:
The 'horizon' axis is drawn in **log scale** in all the plots.

2 Task 1

The python code for the algorithms are almost just as how they were discussed in the lectures, except a few details. I will now go on and describe them a bit.

2.1 The Code

Epsilon Greedy

To decide whether to explore or exploit, a uniform random variable is sampled at each step. If the value sampled is less than ϵ, the function explores(pulls a random arm). Otherwise, it pulls the arm with the maximum P_{avg}. Ties are broken randomly here. At t=0, P_{avg} is taken to be 0 for all arms and the epsilon greedy algorithm starts without any initial round robin exploration.

UCB

The algorithm does a round robin exploration first k steps (where k is assumed to be the number of arms) so that the number of pulls for each arm becomes one. From $(k+1)^{th}$ step onward, the arm with the maximum UCB value[1] is pulled. Here too, the ties are broken randomly.

KL-UCB

The algorithm, like UCB, starts with a round robin exploration. After this, in every time step it finds $q \in$ arms a_i in the bandit instance, such that:

$$\left\| KL(p^t_{a_i}, q), [ln(t) + 3ln(ln(t))]/u^t_{a_i} \right\| \leq 10^{-6} (\approx 0)$$

...(from 2)

using binary search, and pulls the arm with maximum $UCB(= q + p_{avg})$ value, breaking ties randomly. Here, KL(x,y) represents the KL divergence between x and y)
Note that the parameter c in the original equation is taken to be 3. Also, the equation given may violate the inequality, but we do not really care about it given the precision(10^{-6}).

Thompson sampling

At every time step, the algorithm samples from a Beta distribution defined for each arm distinctly with parameters,

$$\alpha = s^t_a + 1 (=\text{reward obtained by arm + 1}), \text{ and } \beta = f^t_a + 1 = u^t_a - s^t_a + 1.$$

[1] $UCB^t_a = \hat{P_a}^t + \sqrt{\frac{2ln(t)}{u^t_a}}$ - Slides

[2] Question 5 in QnA Slides

It then pulls the arm whose sampled value is maximum. There is no method implemented for breaking ties. It simply uses np.argmax for it. This is because the probability of two values being equal is very low, since Beta distribution is continuous and the precision of its discretization is very high.

Now that we have looked at how they work, let see how they perform on the bandit instances.

2.2 The Data

Figures 1,2 and 3 show the plots of 'Regret V/S Horizon' for the instances, i1, i2 and i3 respectively(as mentioned, horizon is in log scale). The algorithms- epsilon greedy, UCB, KL-UCB and Thompson Sampling are compared in each case.

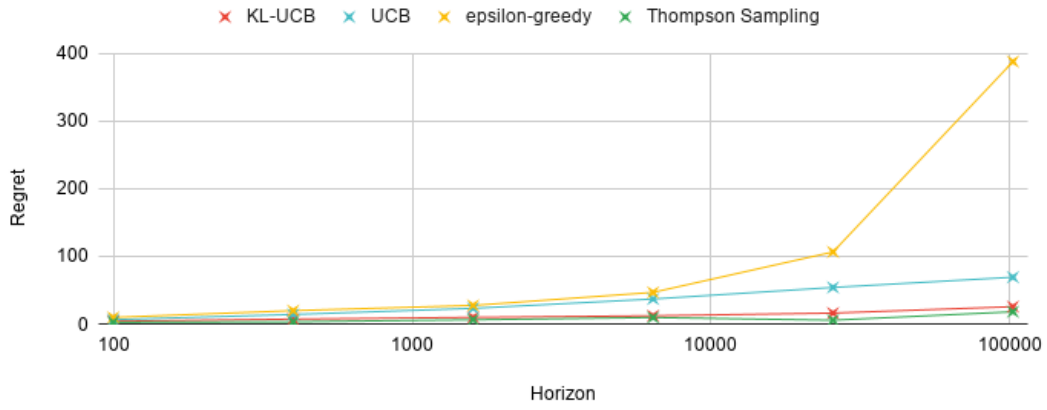

Figure 1: Mean regrets for instance i-1

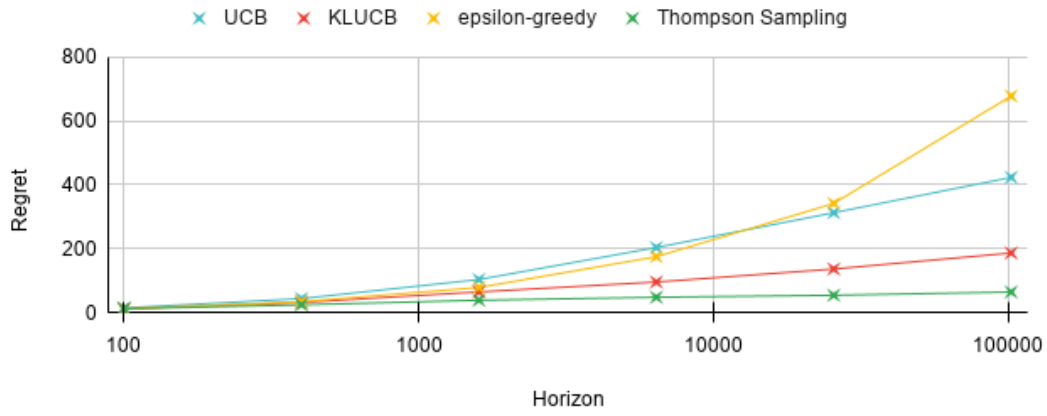

Figure 2: Mean regrets for instance i-2

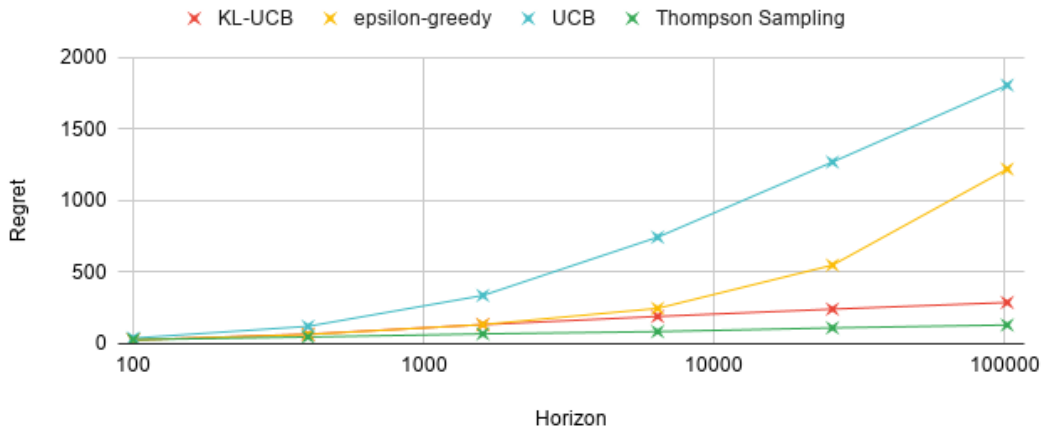

Figure 3: Mean regrets for instance i-3

The plots for KL-UCB and Thompson sampling look linear with respect to the logarithmic 'Horizon axis'. This seems to be consistent with their logarithmic $\mathbb{E}(regret)$[2, 3]. Moreover, we can observe that Thompson sampling performs better than kl-UCB in all instances.

UCB is linear in plots 1 and 2, but, in 3, it performs worse than epsilon-greedy, which is supposed to have a linear regret(slides). However, on fitting a linear curve on epsilon-greedy's plot and a logarithmic curve on UCB's plot gives us the best R^2 value (≈ 0.94 for both cases[4]). It is just that epsilon greedy performs better with $\epsilon = 0.02$ here. This may be because the confidence bounds of the UCB algorithm are not tight enough. We can see that P_a for each arm in i-3 is very close to each other. Hence, it(UCB) may fail to choose the optimal arm. This is confirmed by seeing the performance of KL-UCB on i-3, which clearly overcame the problem and gave lesser regret, since the only difference is that KL-UCB gives tighter confidence bounds. So, given enough time-steps, the UCB would give lower regrets (which is confirmed by the R^2 values I got for the curves).

The plots corresponding to the epsilon greedy algorithm are exponential for all the instances(of course, with respect to log(horizon)). Thus, confirming its linear regret.

3 Task 2

This section explains how I used the 'hint' to get a better performance than the Thompson Sampling algorithm. First, I describe the code and the intuition behind it. Then, following the conventional format of this report, I show its performance via plots of generated data.

[2]The KL-UCB Algorithm for Bounded Stochastic Bandits and Beyond [link]
[3]Analysis of Thompson Sampling for the multi-armed bandit problem [link]
[4]Note that this is not overfitting the data as it gives a good R^2 value for other instances as well. (Hence, the curve generalises well)

3.1 The Code

The hint given to us is a random permutation of the arms in the instance. I have only used a subset of it, namely the P^* value for each instance. So, in the code, I have not shuffled or sorted the arms. Instead just passed a single integer value.

The arm instances are stored in a private variable in a class called arms. So it is not inherited from it. This confirms that the entire code does not use the sequence of instances as it is.

Lets move on to the algorithm now.

Thompson sampling with hint

The algorithm samples from a beta distribution defined for each arm in the exact same way as the Thompson Sampling algorithm does. The only addition to this is that, instead of pulling the arm with the maximum sampled value, it pulls the arm given by:

$$argmax_{a \in instance}(\beta_a^t + exp(-u_a^t KL(\hat{p_a}^t, P^*)))$$

where β_a^t is equal to the value sampled from the arm's Beta distribution. Even here, like Thompson sampling, there is no round robin or method of breaking ties. The idea is to add a bias to the bayesian prior estimate. Since we know the best arm's mean, we can use the KL divergence to measure how different is an arms distribution from the 'ideal' one. KL divergence, scaled, acts as our bias.

Suppose some arms have $P_a^t \geq P^*$ at some time-step. if the real mean of such arms is not equal to P^*, the probability that its mean will remain greater than P^* drops exponentially[5]. If it is, then the second term in the equation keeps increasing for the arm as we continue to sample it.

A way to boost the above method even further would be to add additional arms with $P_{avg} = P^*$, as adumbrated in 3. However, I abstained from using it as I was worried about the legitimacy of changing the bandit instance itself[6].

3.2 The Data

Figures 4,5 and 6 show the plots of 'Regret V/S Horizon' for the instances, i1, i2 and i3 respectively(as mentioned, horizon is in log scale). The Thompson Sampling algorithm and the algorithm which incorporates a hint about the instances are compared in each case.

[5] $P(\hat{x} > \mu + \epsilon = exp(-u_a^t KL(u + \epsilon, u))$- A KL Inequality(Slides)

[6] One could argue that if allowed to change the instances, he/she would make an entirely new one with only one arm with p=P^*

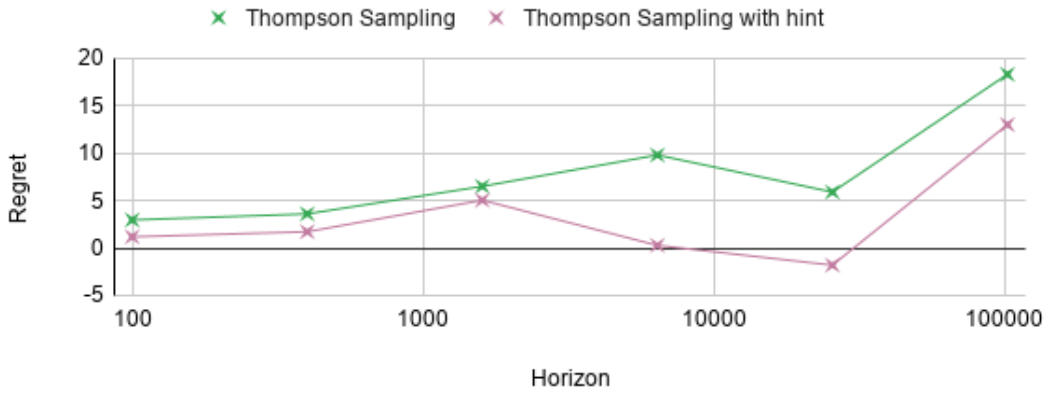

Figure 4: Mean regrets for instance i-1

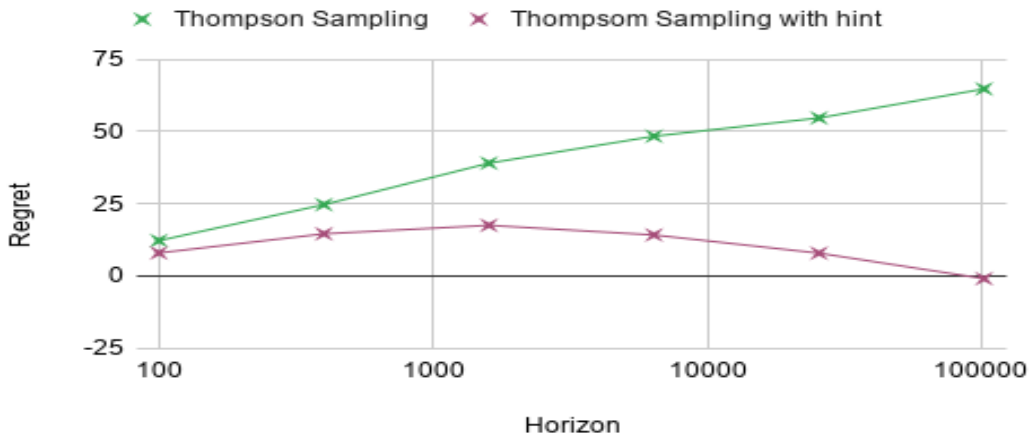

Figure 5: Mean regrets for instance i-2

The algorithm- 'Thomas sampling with hint' clearly does better than Thomas Sampling in all the given instances and horizons. Despite how erratic the former's plot looks, its fits pretty well to a linear curve in all three instances. The algorithm also gives a negative regret even after averaging across 50 seeds, which is quite interesting.

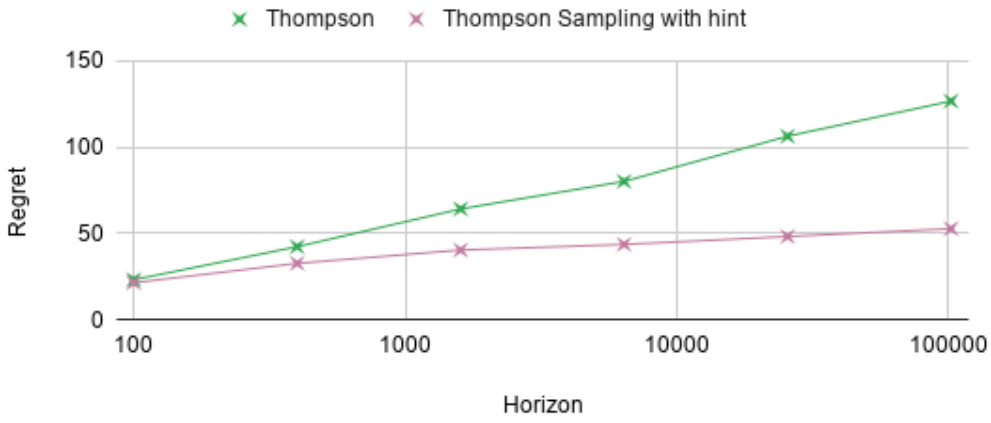

Figure 6: Mean regrets for instance i-3

4 Task 3

From Table 1, we can see that the set {0.001, 0.02, 0.999} satisfies the condition for all the instances, with horizon=102400. With $\epsilon = 0.001$, we are exploring a bit lesser when compared to $\epsilon = 0.02$, and with $\epsilon = 0.999$, we are exploiting very less.

Instance	$\epsilon = 0.001$	$\epsilon = 0.02$	$epsilon = 0.999$
i-1	434.02	387.48	20496.3
i-2	2479.86	677.88	20458.94
i-3	3717.5	1219.2	42275.22

Table 1: Regret means for different epsilons