CS747: Foundations of Intelligent and Learning Agents Assignment-1

Pulkit Paliwal
20D100021

September 10, 2022

1 Task-1

1.1 UCB Algorithm

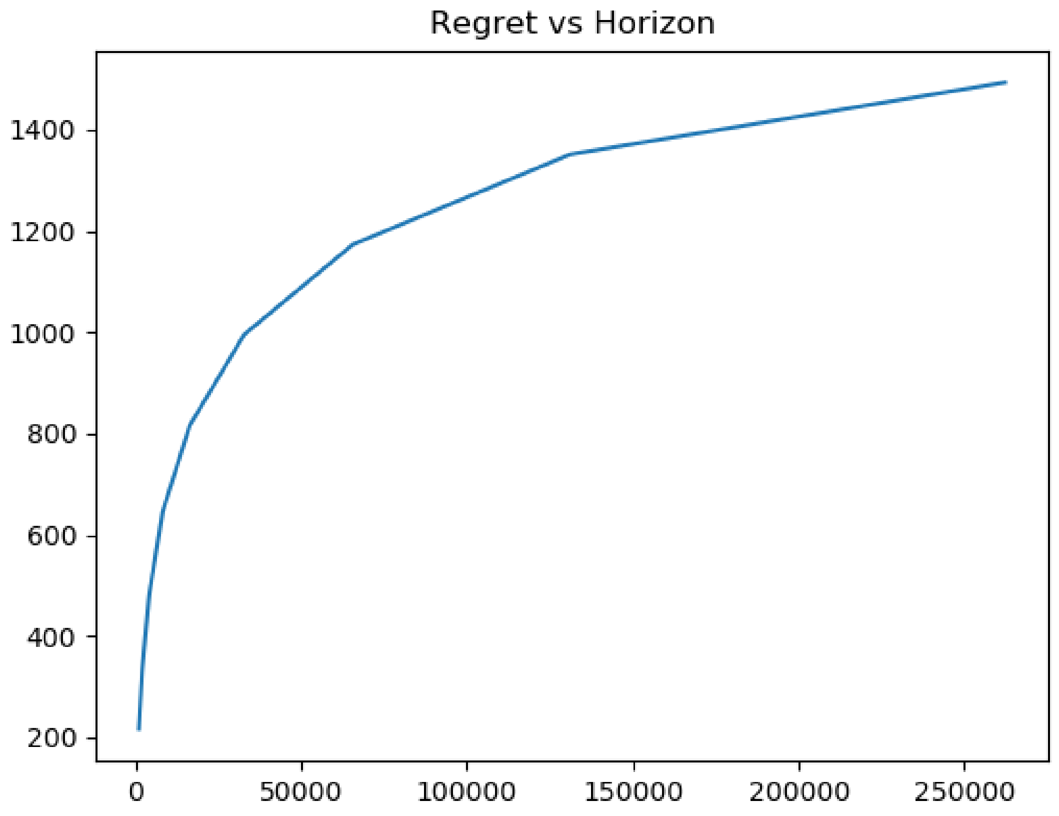

Figure 1: Regret vs Horizon for UCB Algorithm

The UCB algorithm has been implemented in a very straightforward manner. We have two functions: give_pull get_reward. The give_pull function updates the main_count, which is equivalent to t and returns the arm number with the maximum upper concentration bound. The get_reward function calculates the new empirical mean in accordance with the reward obtained, and also calculates the new upper concentration bounds for all the arms.

1.2 KL-UCB Algorithm

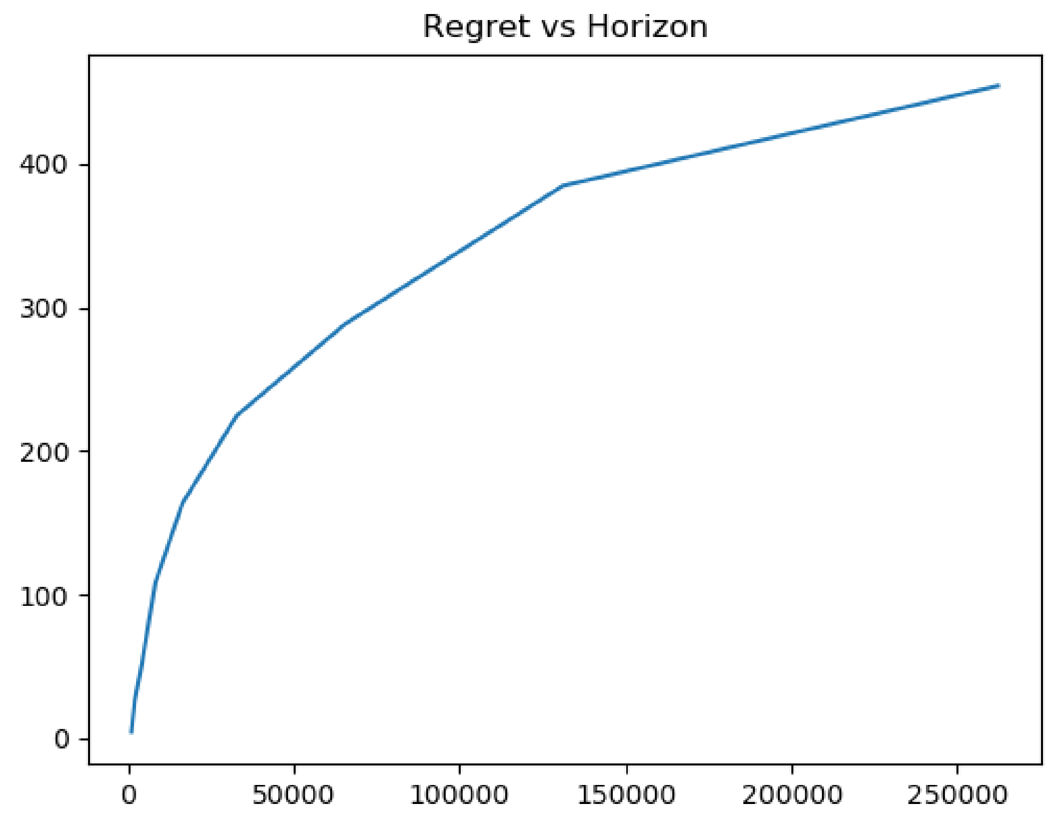

Figure 2: KL-UCB algorithm, Regret Vs Horizon

The KL-UCB algorithm implementation is similar to the UCB algorithm, with the only difference being that instead of upper concentration bounds, we choose our upper bound q to be the maximum value in the interval $[p_a^t, 1]$ such that the KL divergence of p_a^t and q is strictly less than or equal to $\frac{log(t)+3log(log(t))}{u_a^t}$. The implementation involves searching for q for each arm as a solution to the equation $KL(p_a^t, q) = \frac{log(t)+3log(log(t))}{u_a^t}$.

1.3 Thompson Sampling

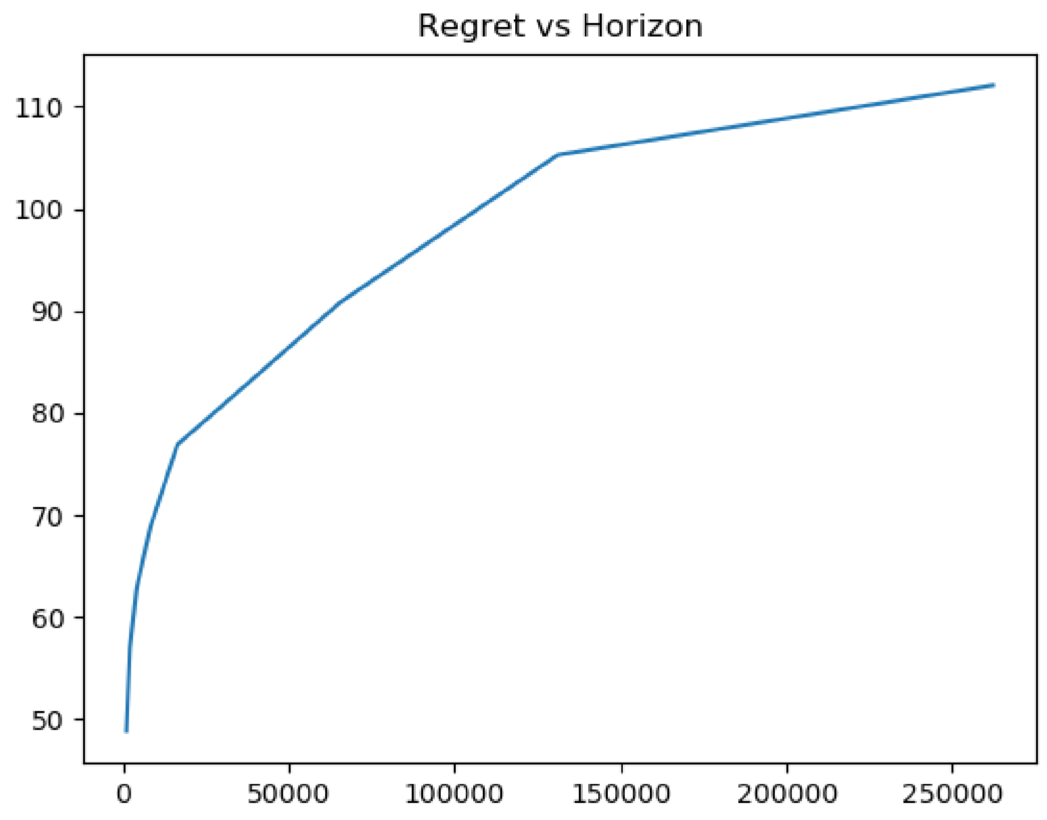

Figure 3: Regret vs Horizon for Thompson Sampling Algorithm

The implementation of thompson sampling algorithm is fairly simple. We define three arrays, s, f, x, which represent the number of successes, failures, and the random value from beta distribution respectively for each arm. The give_pull function samples x for each arm from the beta distribution and returns the index of the arm with the highest sampled value from the beta distribution. The get_reward function updates s and f.

2 Task-2

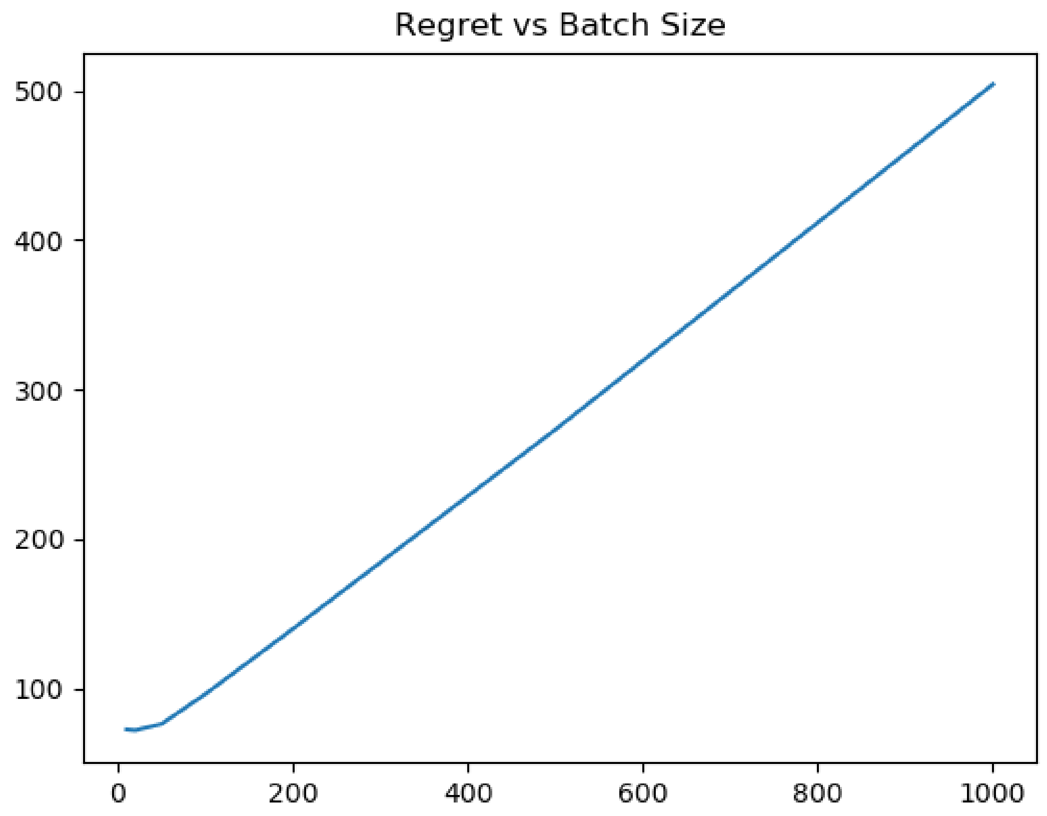

Figure 4: Regret vs Horizon for Batched-Sampling

We define three arrays, s, f, x, which represent the number of successes, failures, and the sampled value from beta distribution respectively for each arm. The give_pull function iteratively samples batch_size number of arms, returning the index of the arm with the maximum value from the beta distribution in each iteration. The get_reward function updates s and f in accordance with the supplied arguments to the function.

3 Task-3

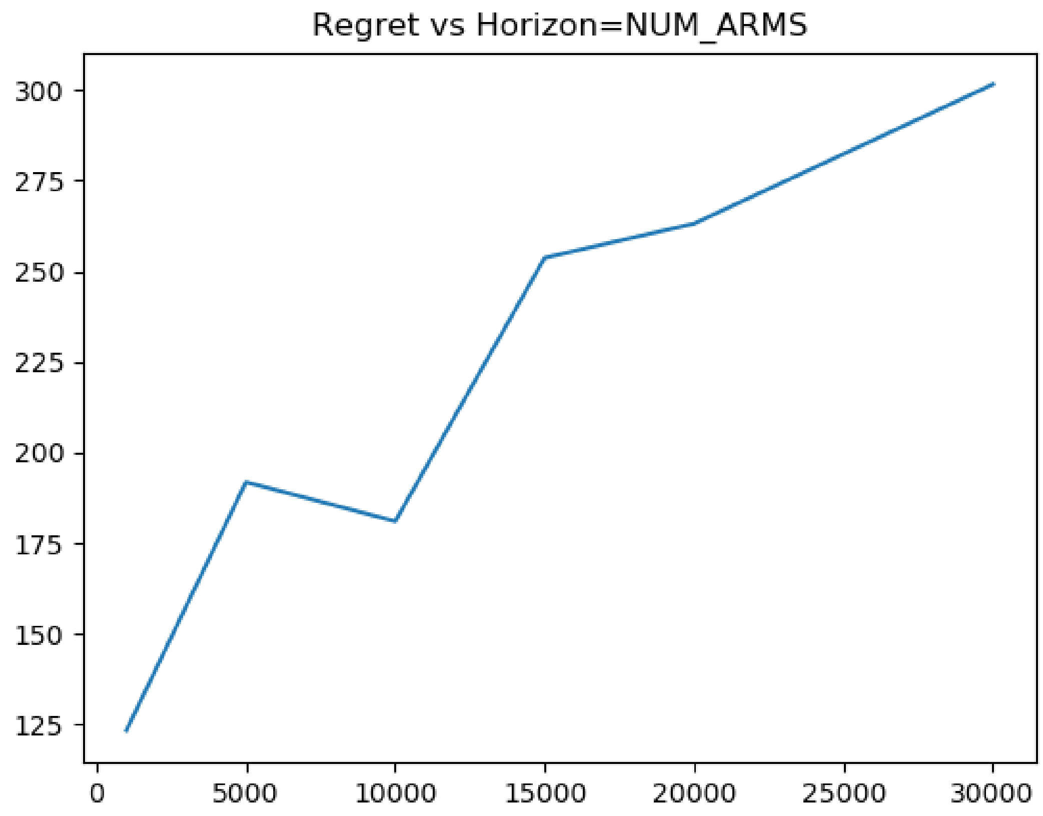

Figure 5: Regret vs Horizon for Many-Arms Algorithm

The implementation for task-3 has been done in a very intuitive manner. We are given that the number of arms is equal to the horizon. Thus, we don't have a lot of scope to actually explore, and hence need to somehow work out an algorithm that will return higher rewards with lower exploration. One such way of doing this could be as follows: We initially sample an arm randomly. Then, if an arm returns a reward of 1, we continue sampling it. If not, we check whether there exist arms with empirical means above a certain threshold. If they do, then we go ahead and sample the arm with the highest empirical mean. We also maintain an array of 'pull-worthy' arms alongside, where we store the indices of the arm that can be pulled. Thus if an arm returns a reward zero and the empirical mean condition is false as well, then, first, we delete the last arm pulled from the list of 'pull-worthy' arms and then randomly sample an arm from the 'pull-worthy' set of arms. For practical purposes, I have chosen the threshold to be 0.96 (fairly high value, but not too high, considering the arm means are distributed randomly but uniformly in arithmetic progression, allowing the possibility to explore and stumble upon arms with sampled empirical means > 0.96).