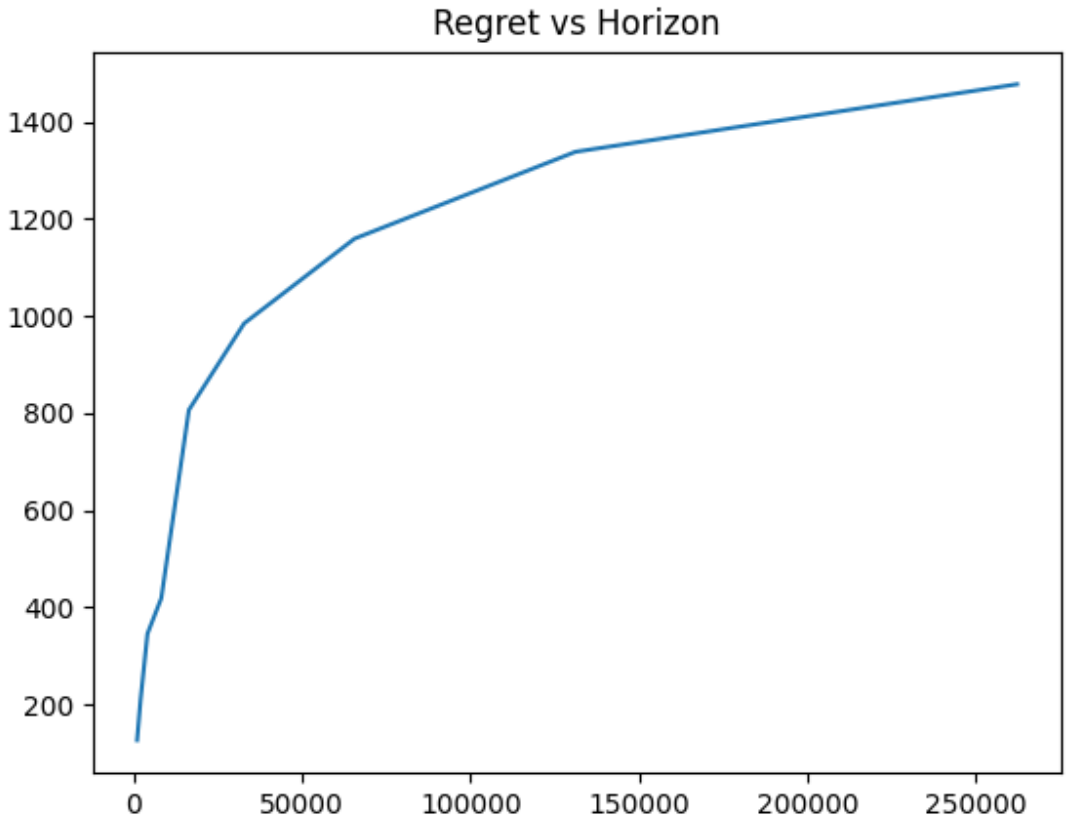
CS747 Programming Assignment 1

Manav Doshi
200100094

Task 1 - Implementing UCB, KL-UCB and Thompson Sampling

Below you can find the graphs (Regret vs Horizon) for the algorithms

- UCB

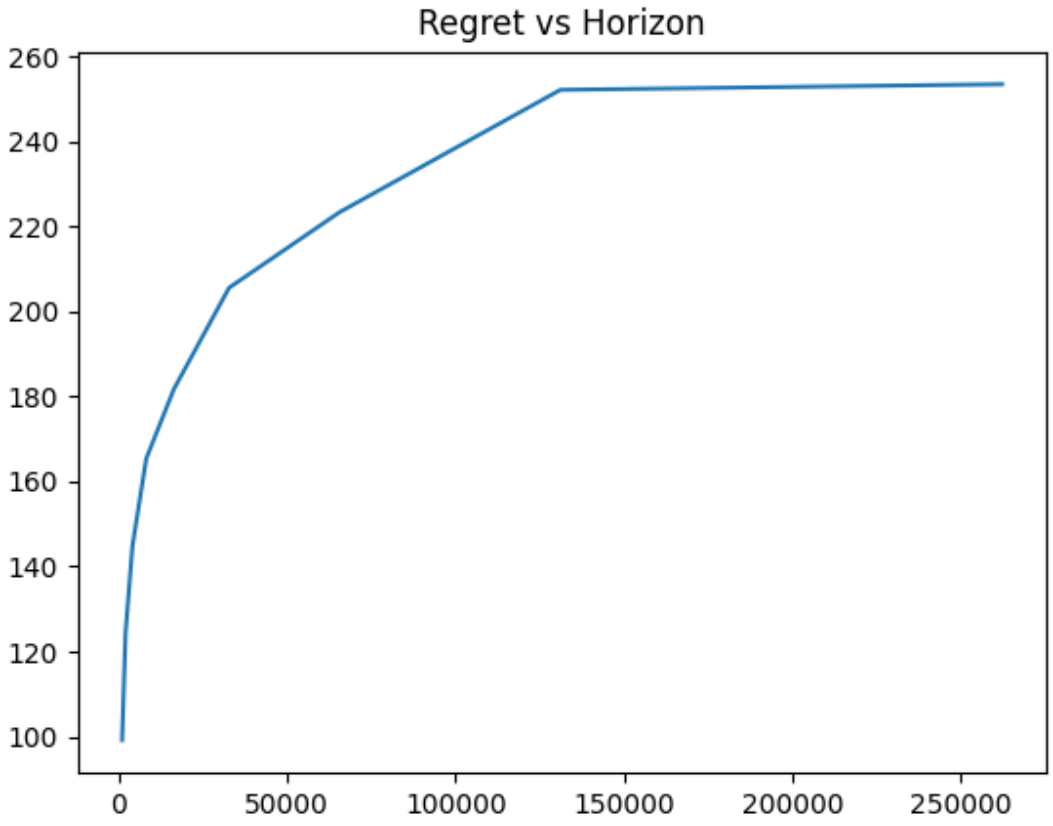


  - Four state variables - mean, counts, time and an array called ucb are defined
  - We select the arm with the maximum Upper Confidence Bound (UCB) which is the method suggested by the paper.

$$\text{ucb}_a^t \stackrel{\text{def}}{=} \hat{p}_a^t + \sqrt{\frac{2\ln(t)}{u_a^t}}$$

- With every arm pull, the mean reward and counts for each arm are updated and the ucb array recomputed for each arm.
- The time step is also increased at every step since the UCB for each arm also depends on the time

- KL-UCB



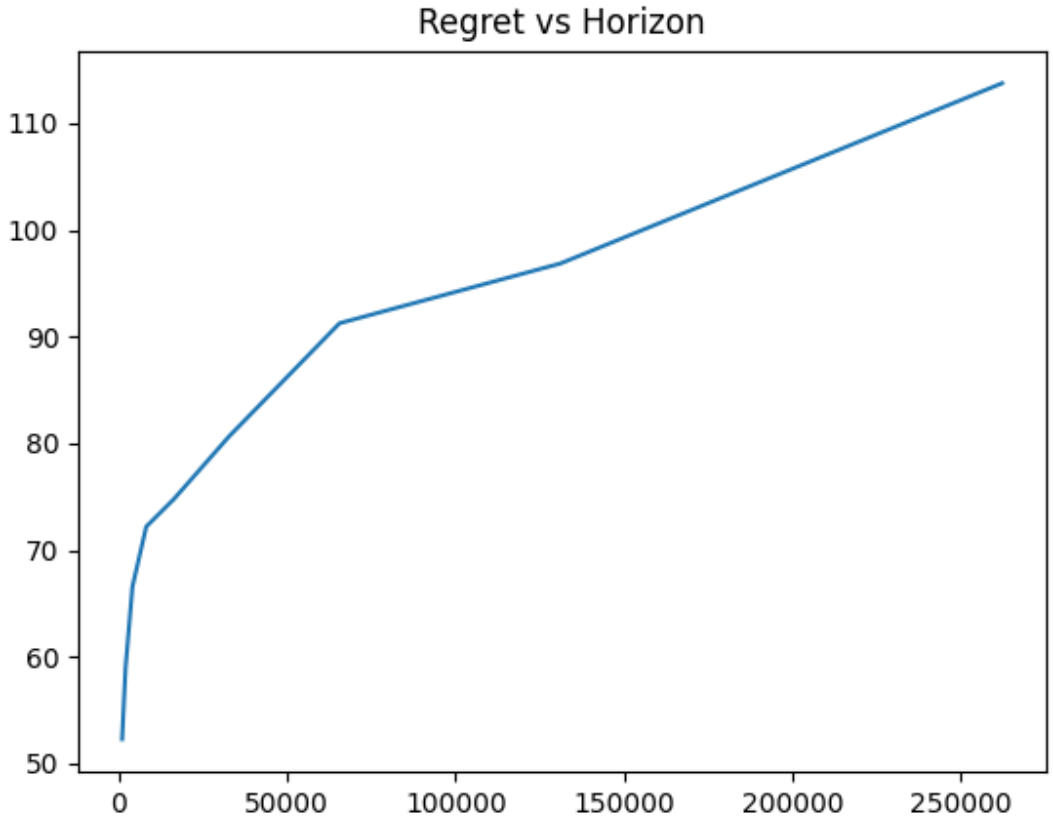
- This algorithm works in a similar way to the UCB algorithm but differs in the way we calculate the upper confidence bound for each arm
- Three helper functions are defined - to calculate the KL divergence of two numbers, to calculate the right hand side time expression in the KL-UCB inequality and a function to find the solution to the inequality

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

- C here is taken to be 3
- The first two functions are quite straightforward but the third function uses kind of a bisection method to calculate the solution to the inequality
- It starts with defining a step variable and checking if the mean and the solution + step satisfies the inequality. If yes, the solution is incremented
- At every step the step size is halved to increase the precision

- This algorithm is similar to the above algorithm after having calculated the the UCB array

- Thompson Sampling

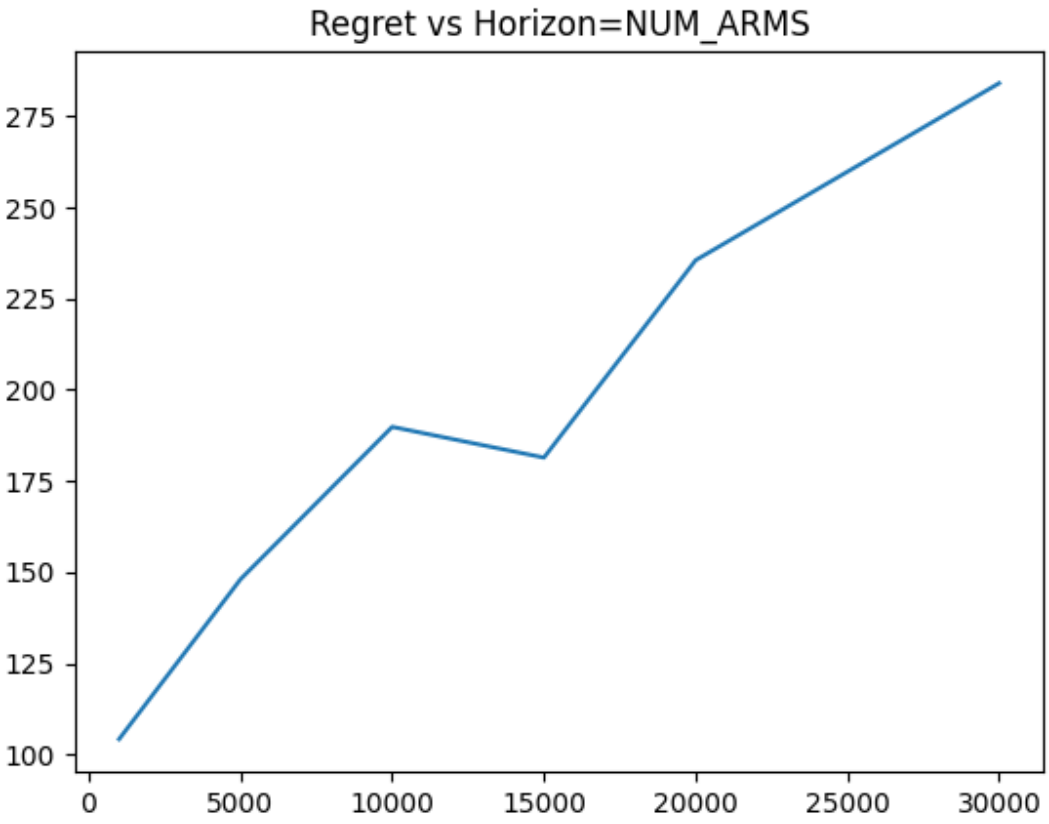


- This algorithm works in a different way compared to the other two above
- Here two additional state variables are defined - arrays called success and failure which store the number of success and failures for each arm
- To select arm to be pulled, a value is sampled from a beta distribution given by Beta(success + 1, failure +1) and the arm with the highest value is pulled
- After pulling, the number of success/failure array is updated to reflect the latest values

Task 2 - Algorithm for batched sampling

- Here since the optimal policy I know of is Thompson Sampling, I used a version of Thompson Sampling to effectively find arm indexes and number of times they should be pulled
- Three state variables were defined - success, failure and samples all numpy arrays of sizes num_arms, num_arms and num_arms*batch_size respectively
- Every arm was first sampled batch_size number of times giving us a num_arms*batch_size array.
- The top batch_size number of arms were pulled every time giving us a variant of Thompson Sampling and it achieves very low regret over all batch sizes
- The get_reward function updates the success and failure array with the number of times the arm gave us a reward of 1 and 0 respectively.
- The graph above makes sense as with a low batch size, we can effectively explore arms before moving on to exploit them whereas with higher batch sizes since we do not know the actual means of the arms, it is difficult to exploit giving us a higher value of regret

Task 3 - Algorithm for when number of arms equals horizon



- In this case, since most of our theories don't apply none of our usual algorithms would work well
- Thinking upon the question, it is very difficult to explore effectively when number of arms equals the horizon - hence the logic behind my code - only exploit
- We now are relieved of the exploit vs explore trade off since only one of them can be done effectively, I move on to constantly exploit at every time step
- State variables index, values and counts are defined in the class
- Since the arms are initially randomly permuted, we start with index = 0 and pull that particular arm
- After having pulled this arm, we update the mean value of the reward given by this arm and check the reward received
- If this particular value of reward is 1, we exploit by pulling this arm again but if it is 0 we check the mean value of the reward given by this particular arm
- This mean value was checked against a parameter (0.98 in this case) and if the mean is greater, then that means it is one of the optimal arms and we should keep pulling it for lower regret values.

- The graph gives regrets of the order of the KL-UCB and Thompson Sampling Algorithms and we can clearly see that it is one of the optimal policies for when exploration is not a feasible option.