Foundations of Intelligent and Learning Agents CS747

Mohit
20D070052

September 2023

1 Student Details

Name:	Mohit
Roll No:	20D070052

2 Task 1

2.1 UCB Algorithm

We know that in UCB algorithm the arm to be pulled is selected by :

$$ucb_a^t = \hat{p}_a^t + \sqrt{2ln(t)/u_a^t}$$

Now in the algorithm, Parameters :
self.counts - An array which counts the number of times each arm was pulled as per it's index
self.values - An array which keeps a track of the rewards that we are getting from each arm
self.empmean - An array which keeps a track of the empirical mean of each arm after every pull given by reward for that arm/ number of times that arm was pulled
self.time - Keeps a track of the time i.e. mainly the total number of pulls till now

Note : We could have removed the empmean and time variables but they help in providing more readability hence they are kept

Now the algorithm is mainly in the givepull function :
Firstly we initiate 2 variables i.e. maxucb to record the maximum ucb for all arms also index will be that arm which has the maximum ucb. Now initially we will sample each arm once so that u_a^t is non zero for all arms after this. After this we have a loop in which we iterate through all the arms and check which arm has the maximum ucb and return the index corresponding to that arm.

Now in the getreward function we update the count of the arm that was pulled, add the reward corresponding to that arm pull, increment the time by 1 and update the empirical mean of that arm.

2.2 Plot

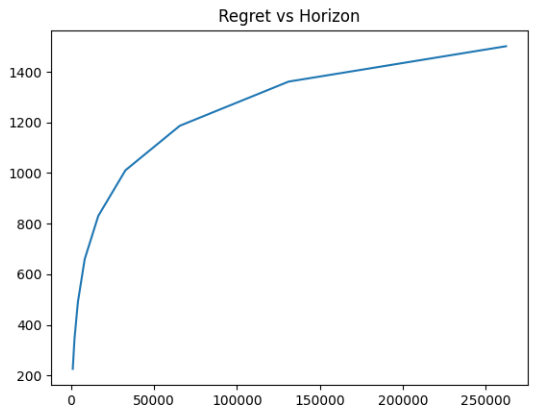

Figure 1: Regret vs Horizon for UCB Algorithm

As expected the regret is logarithmic with respect to the horizon which is much better than the E-Greedy Method.

2.3 KL-UCB Algorithm

We know that in KL-UCB algorithm the arm to be pulled is selected by :

$$ucbkl_a^t = maxq \in [p_a^t, 1] such that (u_a^t)(KL(p_a^t, q) \leq ln(t) + cln(ln(t)), c \geq 3$$

However in class it was told that c should be $\geq$ 3 but in the paper it is recommended to take c = 0 and testcases are also designed according to that so we will also use the same.

Now in the algorithm, Parameters :
self.counts - An array which counts the number of times each arm was pulled as per it's index
self.values - An array which keeps a track of the rewards that we are getting from each arm
self.empmean - An array which keeps a track of the empirical mean of each arm after every pull given by reward for that arm/ number of times that arm was pulled
self.time - Keeps a track of the time i.e. mainly the total number of pulls till now

Note : We could have removed the empmean and time variables but they help in providing more readability hence they are kept

Now in the algorithm is mainly in the givepull function :
Firstly we initiate 2 variables i.e. qmax to know that what is the maximum q for any arm till now also index will be that arm which has the maximum q. Now initially we will sample each arm once so that u_a^t is non zero for all arms after this. Now in this algorithm we have used the fact that KL divergence is an increasing function in the variable q so to make the algorithm optimized we will use binary search to reduce time complexity. After this we have a loop in which we iterate through all the arms and for each arm we check the maximum q for which the condition is satisfied and check if that q is greater than qmax then we update the qmax with this q and index with this index. Also we want to run the algorithm to run atleast once so that's why there is a variable of count to count the number of times the loop has run cause there can be a case in which p_a^t is already equal to 1 so we want to include that case also.

Now in the getreward function we update the count of the arm that was pulled, add the reward corresponding to that arm pull, increment the time by 1 and update the empirical mean of that arm.

2.4 Plot

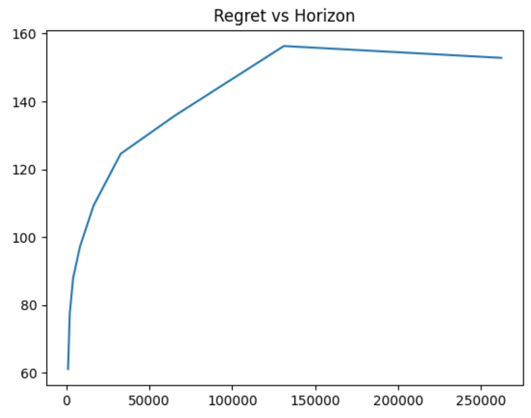

Figure 2: Regret vs Horizon for KL-UCB Algorithm

As expected the regret is logarithmic with respect to the horizon which is much better than the E-Greedy Method. The regret is much better than the UCB algorithm also because KL-UCB gives a much tighter confidence bound than UCB and the regret matches the lower bound. And even the regret decreases because there is a high probability of regret going negative since our expectation is now zero once we reach optimal convergence. The regret seems to be linear in the later part but that is because of the step size of the horizon we are using.

2.5 Thompson Sampling Algorithm

We know that in Thompson Sampling Algorithm the arm to be pulled is selected by :

Now in the algorithm, Parameters :
self.counts - An array which counts the number of times each arm was pulled as per it's index
self.values - An array which keeps a track of the rewards that we are getting from each arm

Now in the algorithm is mainly in the givepull function :
Firstly we initiate 2 variables i.e. the max sample that we have recieved from all arms and index which stores the index of that arm for which we got the max sample. Now initially we will sample each arm once so that . After this we have a loop in which we iterate through all the arms and check which arm has the maximum ucb and return the index corresponding to that arm.

Now in the getreward function we update the count of the arm that was pulled and add the reward corresponding to that arm pull.

2.6 Plot

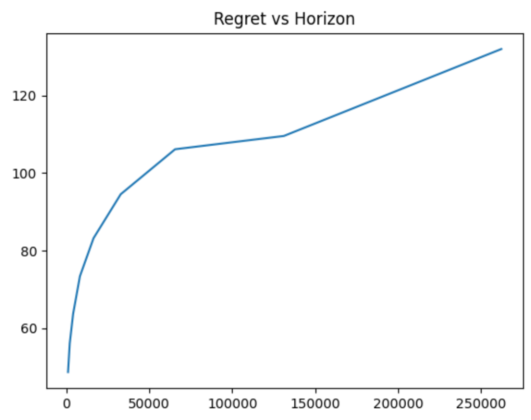

Figure 3: Regret vs Horizon for Thompson Algorithm

As expected the regret is logarithmic with respect to the horizon which is much better than the E-Greedy Method. The regret is much better than the UCB algorithm also because Thompson gives a much tighter confidence bound than UCB and the regret matches the lower bound. The regret seems to be linear in the later part but that is because of the step size of the horizon we are using.

We can also observe that the regret for Thompson method is smaller than the KL-UCB hence we will use this in further tasks also like 3 and 4.

3 Task 2

This task was divided into 2 parts which are explained below:

3.1 Task 2a

In this task we have increased the p2 from 0 to 0.9 in steps of 0.05 over a horizon of 30,000. The results obtained are as listed below in the next sections :

3.1.1 Plot

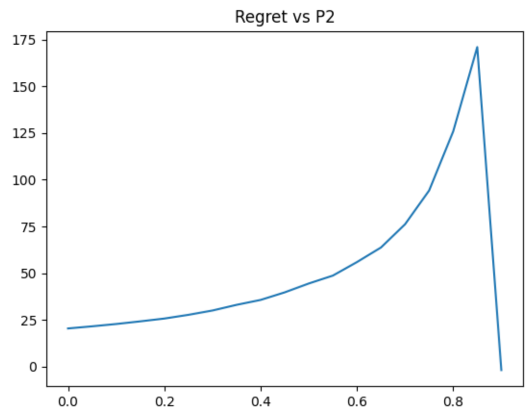

Figure 4: Regret vs P2 for UCB Algorithm

3.1.2 Observations and Reasoning

We observe that the regret continues to increase upto a certain point and then abruptly falls to zero. The regret when both probabilities are equal is expected to be zero. Since both the arms have equal probability of reward it doesn't matter which arm we pick so regret has to be zero. The regret continues to increase upto 0.85 and has to be zero at 0.9 because of the step size else it would have increased after 0.85 also.

3.2 Task 2b

In this task we have increased the p2 from 0 to 0.9 in steps of 0.05 over a horizon of 30,000 but also we are increasing p1 such that p1 - p2 = 0.1 remains constant. The results obtained are as listed below in the next sections :

3.2.1 Plots

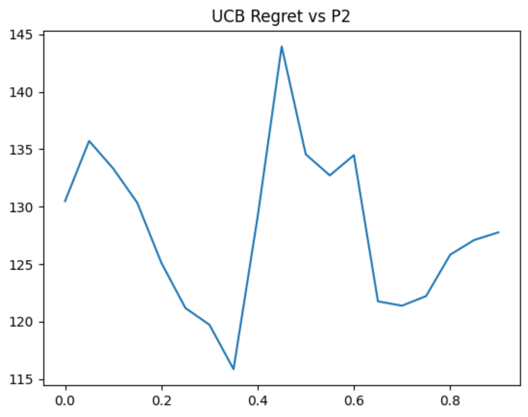

Figure 5: Regret vs P2 for UCB Algorithm

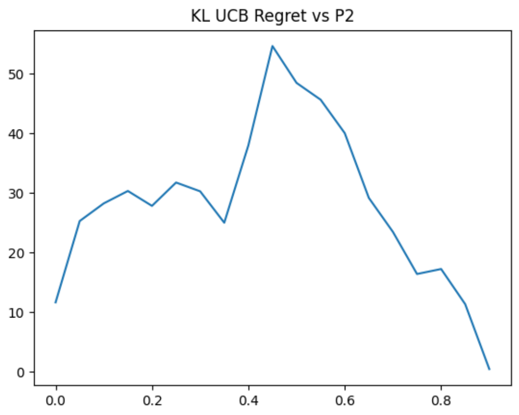

Figure 6: Regret vs P2 for KL-UCB Algorithm

3.2.2 Observations and Reasoning

- Firstly we can observe that the regret for KL-UCB is less than half of the UCB algorithm and this is expected because as studied in class the KL-UCB gives a much tighter confidence bound than UCB.
- For both of these algorithms we can observe that the regret maximizes when p2 goes from 0.35 to 0.45 i.e. when p1 goes from 0.45 to 0.55.

4 Task 3

In this task we were a given that an arm can also give a faulty output.

4.1 Algorithm

In this question also we can directly use the Thompson Sampling Method. How the algorithm is implemented is already answered in Task 1. The explanation behind the algorithm is as given below :

4.2 Plot of Reward vs Horizon

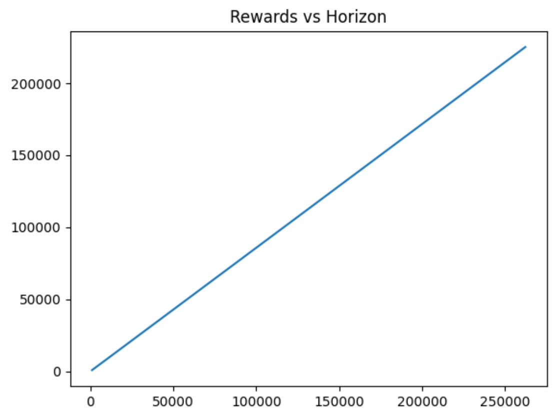

Figure 7: Reward vs Horizon for Faulty Bandits Algorithm

4.3 Reasoning Behind Algorithm

Here it is given that the fault is equal for all arms that if an arm is pulled it gives faulty output. Therefore we can use Thompson Sampling here also because irrespective of the arm that is pulled fault probability is equal hence ideally we would like to follow the optimum method only so that if the fault is not detected then at least we are picking the right arm and if a fault is detected then it will not matter which arm is being pulled hence we will be able to get the best reward through thompson sampling. Also there is one rare case in which this algorithm can fail and that is when the arm with maximum probability of reward (greater than 0.5 i.e. the fault reward) gives faulty reward everytime and thus affecting its beta distribution to make it below a less optimal arm however this is quite rare because other arms also will have the same problem and hence net effect can be ignored and we can assume that the arm with maximum probability will still be largest. This algorithm can fail if each arm has different probability of faults but here it is not the case hence the algorithm works and reward is also linear with regret.

5 Task 4

In this task we were dealing with 2 bandit instances at once and arm for each of the bandit could be pulled randomly.

5.1 Algorithm

In this question also we can directly use the Thompson Sampling Method. How the algorithm is implemented is already answered in Task 1. The explanation behind the algorithm is as given below :

5.2 Plot of Reward vs Horizon

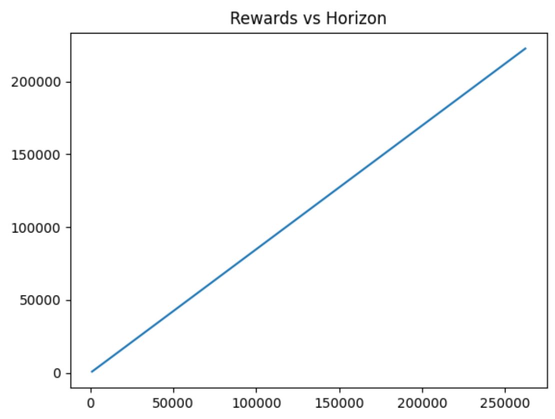

Figure 8: Reward vs Horizon for Multi-Bandit Algorithm

5.3 Reasoning Behind Algorithm

In Thompson sampling we take a random sample from the beta distribution of each arm, since the probability of selecting each arm is 0.5 due to uniform choice of bandit. So if we combine both of these bandit instances we can have a joint probability distribution of each arm and select the sample of maximum value after this it won't matter which bandit instance is chosen cause we are taking both of these instances together. Otherwise we could have used 2 separate thompson sampling arrays for each bandit instance, select samples from beta distributions, and sort both of these arrays in increasing order along with the arm for which they are observed. Now we can select that arm which appears in both of these arrays in the initial part so that whichever bandit gets selected we get better reward. However this approach seemed computationally heavy and directly using the first method works with a linear reward hence that method was used.