Assignment 1

C S747 : Foundations in Intelligent Learning Agents
Indian Institute of Technology Bombay

Amritaansh Narain
200100022

---

1 Task 1

1.1 UCB Algorithm

Each instance of UCB algorithm includes the UCB value, empirical mean, number of pulls of each arm till current time and the time itself. UCB algorithm works by pulling the arm with the maximum UCB value at a given time, $t \geq$ number of arms. UCB value of an arm at a given time is defined as follows

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

Initially from time $t = 0$ till $t = numberofarms - 1$, we pull each arm once to get some initialization for the empirical mean. The above condition of pulling was implemented using conditionals and $np.argmax()$. Once we get the reward for an arm, we update the empirical mean of the arm which was pulled currently. We also update the UCB values for every other arm, as the time has increased hence the UCB value for each arm will change. This was implemented by updating UCB value for each arm by iterating over the arms via loop and updating empirical mean of arm when current arm is the arm pulled. On running the 'simulator.py' over UCB we get the below graph, we can see it converging after around 1400 regret.

1.2 KL-UCB Algorithm

Each instance of KL-UCB Algorithm includes KL-UCB value, empirical mean, number of pulls of each arm. KL-UCB works by pulling the arm with largest KL-UCB value at a given time t, $t \geq$ number of arms. KL-UCB value of an arm at given time t is defined as follows,

$$kl - ucb_a^t = max\{q \epsilon [\hat{p}_a^t, 1] | u_a^t \cdot KL(\hat{p}_a^t, q) \leq ln(t) + c \cdot ln(ln(t))\}, c \geq 3$$

From $t = 0$ to $t =$ number of arms $- 1$, I pull each arm once to get an initialization for the empirical mean. Code for pulling the arm was implemented using conditionals on time and $np.argmax()$ over kl-ucb values. Once we get the reward for the pulled arm, we iterate over all the arms and update the kl-ucb value for each arm, and also update the empirical mean for the arm which was just pulled. To update the KL-UCB value of each arm, I use a binary search to find the appropriate q value, as KL divergence function is an increasing function. I define an upperbound as $(ln(t) + c \cdot ln(ln(t) + 1e - 9))$, $1e - 9$ is added at various points along the KL-UCB code because we don't want zero to be inside log function or in denominator, hence we add a very small value to make it non-zero. Hyperparameter c, $c \geq 3$ is taken to be 3. Left index l, and right index r is defined as l = empirical mean $+1e - 9$ and $r = 1 - 1e - 9$ and run binary search over it until $(r - l)$ less than some tolerance and this tolerance is defined as 0.05. In the while loop for binary search, we define mid as middle element between left and right index. When $u_a^t \cdot KL(\hat{p}_a^t, mid) \leq$ upperbound, we update left index to mid, else we update right index to middle element. Eventually this converges till difference between right and left end value 0.05. This value is the new KL-UCB value for the arm. On running KL-UCB over simulator.py we get below graph, with regret going till 200, significantly better than UCB Algorithm.

1.3 Thompson Algorithm

Each instance of Thompson Algorithm maintains the number of successes and failures of each arm at the pulls it has had. When pulling an arm, we sample a value for each arm assuming a beta distribution defined by the number of successes and the number of failures it has had. Then we pull the arm which gave the largest sampled value. Essentially beta distribution of an arm is a distribution about our belief in the actual mean of an arm. When we

((a)) UCB Algorithm

((b)) KL-UCB Algorithm

((c)) Thompson Algorithm

get the reward for an arm, we update the number of success or number of failures of the arm, which in turn would update the distribution of each arm when call the pull function. Below is the graph for the result over simulator.py. We can see that this has the lower regret than both UCB and KL-UCB. Graph doesnt seem like it is converging unlike KL-UCB which seemed like it had converged.

2 Task 2

2.1 Idea

Task states that we are allowed to pull in batches, hence at given time instant, I need to give the instructions regarding which arms to pull for the next batch. The pulls for the next batch are to be stated as number of pulls for each arm instead of stating which arm to pull sequentially. Utilizing the above fact, I concluded that each suggested pull for the batch is independent of any other pull suggested for the batch. Hence I thought of using probabilistic prediction algorithm instead of a deterministic one that is, Thompson Sampling algorithm. Using the rewards obtained till time T, I record successes and failures for each arm. To predict pulls for next batch, for each time instance, I sample value for each arm from the beta distribution defined by number of successes and number of failures for the arm. At each time instance, I pull the arm with maximum sampled value. I keep a record of the number of pulls for each arm and return 2 lists, where first list states the arm with non zero pulls and second list states the number of times I pull the arms in first list. Below is the graph of regret vs batch size for the result over 'simulator.py' file.

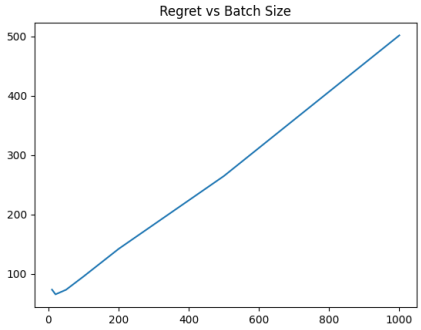

Figure 2: Task 2

2.2 Observations

Graph has a linear profile that is as the batch size increases the regret increases because we are predicting for a larger number of arms. It is not possible to get sub-linear profile with batch size because we are not getting any feedback while suggesting pulls hence the regret can only increase when pulling more and more number of arms without any feedback. Slope of this graph is estimated aroung 0.5, i.e. for batch size of n regret is $\frac{n}{2}$. In the worst case, for the worst algorithm worst regret would be $n \cdot (1 - 0)$ hence our algorithm performs significantly better.

3 Task 3

The task says that we are allowed the number of pulls equal to the number of arms, also we are given that mean reward of each arm is an arithmetic progression from 0 to $1 - \frac{1}{\text{number of arms}}$ and minimize total regret. The first thing which

came to my mind was the optimal algorithm KL-UCB, however both KL-UCB and UCB need an initialization over the empirical mean and for initialization number of pulls need to be done equal to number of arms but that would fail because it would start exploiting only after initialization. So the next idea was implementing epsilon greedy since it doesn't need any initialization over empirical mean. The result was very bad since I was not using the fact that mean rewards are in arithmetic progression. So I made a prior over empirical mean in epsilon greedy algorithm such that, empirical mean of arms are the values in AP. Hence this algorithm is same as epsilon greedy but with initial empirical mean of each arm is $0, \frac{1}{n}, \frac{2}{n}, ..., 1 - \frac{1}{n}$. This gave very good results and graph over simulator as shown here.I believe why the results improved a lot was because I gave a prior distribution of mean rewards, hence decreased the solution space hence gave much lower regret than before. Although we don't know which arms have what mean reward, introduction of this prior at least uses the provided information in decreasing the solution space. So the algorithm is the basic epsilon greedy with prior over empirical means. The empirical mean of i^{th} arm is set as $\frac{i}{n}$, n is the number of arms. For pulling arm I chose a random arm with probability epsilon and chose the arm with highest empirical mean with probability $(1 - epsilon)$. By hyper-parameter tuning, I concluded that a very low epsilon value (0.0003) is getting good results compared to higher epsilon values, so essentially we try to exploit more than explore. Every time we get a reward, I update the value of empirical mean.

Figure 3: Task 3

3.1 Observation

Graph shows that the regret is sub-linear with respect to the horizon i.e. despite increasing horizon the regret although increasing is not increasing linearly with horizon size. Epsilon greedy alone has linear regret vs horizon profile because it tends to keep accumulating regret and never converges to the optimal arm. Epsilon greedy with the given prior over the problem shows regret vs horizon profile as shown. Slope of regret vs horizon seems to keep on decreasing until the horizons simulator runs over.