Programming Assignment 1

Name: Sarvesh Vikas Gharat
Roll Number: 22D1593

September 9, 2022

I. Task 1

A. ϵ Greedy

In ϵ Greedy algorithm, the algorithm samples random arm with probability "ϵ" and the arm having best empirical mean with probability "$1-\epsilon$". As the algorithm doesn't satisfies condition "Greedy in the limit", it won't give us sub-linear regret.

Figure 1: Regret vs Horizon for ϵ Greedy

The same can be seen from 1. We see that as time increases, regret increases in linear fashion.

B. UCB

In UCB, at time t for every arm a, we define ucb_a^t as

$$\text{ucb}_a^t = \hat{p}_a^t + \sqrt{\frac{2\ln(t)}{\mu_a^t}} \tag{1}$$

In above equation $\hat{p}_a^t$ is empirical mean of arm a, and μ_a^t is the number of times arm a has been sampled. At every time for every arm, we calculate ucb_a^t. Based on the values of ucb_a^t, the algorithm samples the arm for which the value of ucb_a^t is maximum.

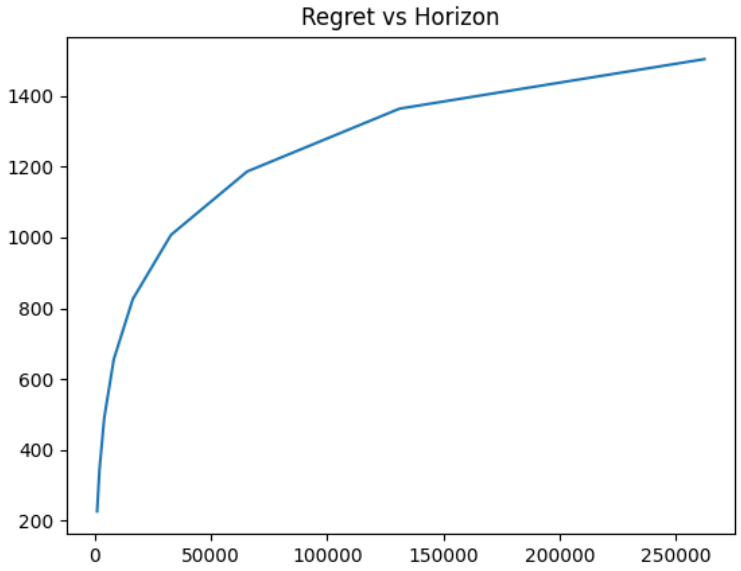

Figure 2: Regret vs Horizon for UCB

This algorithm achieves regret of $O(log(T))$ as seen in Figure 2.

The algorithm for the same is given in Algorithm 1

Algorithm 1 : UCB

Symbol Definition:

1. t: t^{th} time step
2. Horizon: Total Number of Time steps for which algorithm can sample arms
3. Number of Arms: Total Number of Arms
4. Values: Number of times arm returns 1
5. Counts: Number of times arm is sampled
6: Empirical Mean: Ratio of Values and Counts

Give Pull function

Start
if t is less than Number of Arms **then**
    return arm having t as arm index {This is used for round robin}
else
    Calculate Empirical Mean {We don't need for loop as we do array operations}
    Calculate UCB using formula from Equation 1
    return arm with maximum value of UCB
end if
Stop

Get Reward Function

Start
Increase count of arm by 1
if Reward is 1 **then**
    Increase Value of arm by 1
end if
Stop

C. KL-UCB

In KL UCB, at every time t for every arm a we define ucb-kl$_a^t$ as

$$\text{kl-ucb}_a^t = \max\{q \in [\hat{p}_a^t, 1]\} \text{ s.t.} \mu_a^t \times KL(\hat{p}_a^t, q) \leq ln(t) + c.ln(ln(t)), \text{ here } c \geq 3 \quad (2)$$

where KL is defined as

$$KL(p, q) = p.ln\left(\frac{p}{q}\right) + (1-p).ln\left(\frac{1-p}{1-q}\right) \quad (3)$$

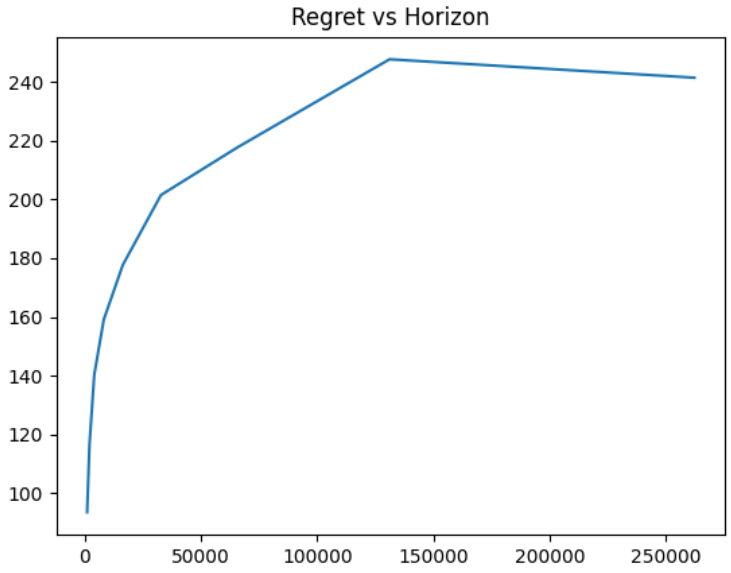

Figure 3: Regret vs Horizon for KL-UCB

KL-UCB in general gives tighter bound than UCB. Even in this case, the algorithm achieves regret of $O(log(T))$ as seen in Figure 3

For this sub task, we define 2 helper functions, KL and BinarySearch which can be seen in Algorithm 2

With help of helper functions as given in Algorithm 2, we define the algorithm for KL-UCB as given in Algorithm 3

Algorithm 2 : Helper Functions for KL-UCB

KL(p,q)

Start
if p == q or q == 0 or q == 1 **then**
    return 0
else
    Calculate KL value from formula shown in Equation 3 with some threshold added in ln
    return KL
end if
Stop

BinarySearch(p, thresh)

Start
Initialise low = p and high = 1
Calculate middle value between low and high
while absolute value of High - Low is less than 10^{-3} **do**
    Initialise q = middle
    if absolute value of KL(p, q) - thresh is less than 10^{-6} **then**
        Break
    else if KL(p, q) is less than thresh **then**
        Change low to middle + 10^{-3}
    else
        Change high to middle - 10^{-3}
    end if
    Calculate middle value between updated low and high again
end while
return middle
Stop

Algorithm 3 : KL-UCB

Symbol Definition:

1. t: t^{th} time step
2. Horizon: Total Number of Time steps for which algorithm can sample arms
3. Number of Arms: Total Number of Arms
4. Values: Number of times arm returns 1
5. Counts: Number of times arm is sampled
6: Empirical Mean: Ratio of Values and Counts

Give Pull function

Start
if t is less than 2 times the Number of Arms **then**
    if t is less than the Number of Arms **then**
        return arm having t as arm index {This is used for first round robin}
    else
        return arm having index as Number of arms - t {This is used for second round robin}
    end if
else
    Calculate Empirical Mean {We don't need for loop as we do array operations}
    for arm in Number of Arms **do**
        Define threshold as $ln(t) + c.ln(ln(t)),$ here $c \geq 3$
        Find $kl - ucb_a^t$ using Binary Search with attributes as empirical mean of the arm and threshold
    end for
    return arm with maximum value of $kl - ucb_a^t$
end if
Stop

Get Reward Function

Start
Increase Counts of arm by 1
if Reward is 1 **then**
    Increase Values of arm by 1
end if
Stop

D. Thompson Sampling

In Thompson sampling at every time t, we calculate s_t^a i.e number of success and f_t^a i,e number of failures for every arm.
Further for every arm we draw a sample $x_t^a \sim \text{Beta}(s_t^a + 1, f_t^a + 1)$
The algorithm further samples arm a for which x_t^a is maximal

Figure 4: Regret vs Horizon for Thompson Sampling

From Figure 4, we see that the algorithm achieves sub linear regret.
The algorithm of the same can be seen in Algorithm 4

Algorithm 4 : Thompson Sampling

Symbol Definition:

1. t: t^{th} time step
2. Horizon: Total Number of Time steps for which algorithm can sample arms
3. Number of Arms: Total Number of Arms
4. Values: Number of times arm returns 1
5. Counts: Number of times arm is sampled
6: Beta: Random Sample drawn from Beta Distribution

Give Pull function

1: Start
2: **for** arm in Number of Arms **do**
3: Initialize Beta of arm as sample randomly drawn from Beta distribution with attributes values of arms + 1 and counts of arm - values of arm + 1
4: **end for**
5: **return** arm with maximum value of sample drawn from Beta distribution
6: Stop

Get Reward Function

1: Start
2: Increase Counts of arm by 1
3: **if** Reward is 1 **then**
4: Increase Values of arm by 1
5: **end if**
6: Stop

II. Task 2

In Task 2, we need to provide an algorithm which will sample arms in batch. This algorithm is divided into two functions.
The first function returns indices of arms to be pulled along with their respected frequencies in 2 separate arrays. To do this the algorithm draws random b samples from beta distribution for every arm as we did in "Thompson Sampling", here b is the batch size. Once drawn, it finds out the index of arm having maximum values.

For example consider we have 5 arms and batch size of 7. In Table given below $a_1, a_2, \ldots, e_6, e_7$ are samples drawn from Beta distribution (attributes of Beta are same as Thompson Sampling)

arm	sample 1	sample 2	sample 3	sample 4	sample 5	sample 6	sample 7
arm 1	a_1	a_2	a_3	a_4	a_5	a_6	a_7
arm 2	b_1	b_2	b_3	b_4	b_5	b_6	b_7
arm 3	c_1	c_2	c_3	c_4	c_5	c_6	c_7
arm 4	d_1	d_2	d_3	d_4	d_5	d_6	d_7
arm 5	e_1	e_2	e_3	e_4	e_5	e_6	e_7

Hence, here the algorithm gives an array which has $(\text{argmax}_1(a_1, \ldots, e_1) \ldots \text{argmax}_7(a_7, \ldots, e_7))$.
Further, we find out unique values in this array along with it's frequencies. Both of these arrays i.e array of unique values and array of frequencies is further returned by algorithm.
In short here we make use of "Thompson Sampling" but in batches.

As we know, the second function receives a dictionary with arm number and the rewards (0 or 1) which it has received in form of array. Hence, we need to use this information to update the parameters which we use in Beta distribution as used in Function 1.
Hence to do that, we first separate the dictionary into arm number and array of it's rewards. Further, to modify the counts (number of times, the arm was pulled) and the values (number of times, the arm gave reward 1), we calculate length of the reward array and sum of the same to modify the values of counts and values respectively.

The algorithm of the same can be seen in Algorithm 5

Algorithm 5 : Task 2

Symbol Definition:

1. n: Total Number of Arms
2. s_t^a: Total Number of success in arm a i.e reward = 1
3. f_t^a: Total Number of failures in arm a i.e reward = 0
4. b: Batch Size
5. counts$_a$: Total number of times arm a is sampled
6. values$_a$: Total number of rewards(1's) arm a has got

Give Pull function

1: Start
2: Define empty list X
3: **for** a in range n **do**
4: Randomly draw b samples from Beta$(s_t^a + 1, f_t^a + 1)$
5: Append list of b samples to X
6: **end for**
7: Get indices of maximum value and store it in array y {Please refer the table example which is shown above}
8: **return** unique values and there corresponding frequencies from array y
9: Stop

Get Reward function

1: Start
2: Store arm number and it's corresponding reward for all arms in 2 separate lists
3: **for** a in range length of first list i.e list containing arm number **do**
4: Add length of sublist corresponding to arm from list 2 i.e list containing sublists of reward to counts$_a$ of the corresponding arm.
5: Add sum of sublist corresponding to arm from list 2 to values$_a$ of the corresponding arm.
6: **end for**
7: Stop

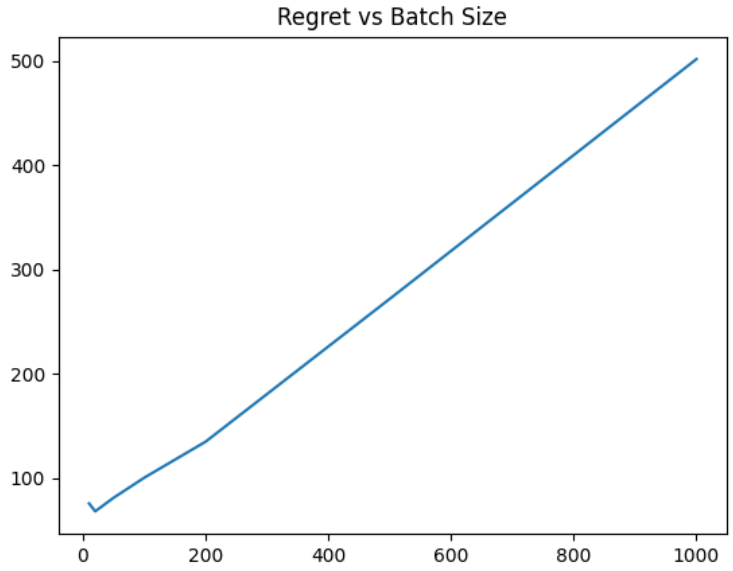

Figure 5: Regret vs Batch Size for task 2

From Figure 5, we see that as batch size increases, initially the regret decreases but later it starts increasing. This task can also be considered as pulling arms from t = 0 to batch size at once, without looking at the success and failures happened in intermediate time i.e time from 0 to batch size. As we increase the batch size we can consider time step to be increased and hence there's lack of information for that particular duration of time which makes the regret higher as we increase the batch size.

III. Task 3

In this task, we are told that the number of arms is equal to number of horizons. Hence, here there is no probability of exploring and than exploiting.
Therefore, in this task we initially sample an arm randomly and go on sampling it until it gives first 0. At that instant, we check whether the empirical mean of that particular arm is greater than some threshold value. If so, than we repeat this process, if not than we sample a random arm again and repeat the process.
The important reason behind not removing the arm from set of all possible arms after it's empirical mean being less than the threshold is there are chances that the optimal or sub-optimal arms with very high mean may give it's first 0 at first initial instance wherein it fails to cross the threshold value and removing such arm won't be a good decision.
The algorithm of the same can be seen in Algorithm 6

Algorithm 6 :Task 3

Symbol Definition:

1. Number of Arms: Total Number of Arms
2. Values: Number of times arm returns 1
3. Counts: Number of times arm is sampled
4: Empirical Mean: Ratio of Values and Counts

Give Pull function

```
Start
Define threshold to be 0.95
if It's the first pull then
    return Arm randomly
else
    for Arm in Number of Arms do
        if Count of particular arm is not 0 then
            Calculate Empirical Mean
            if Empirical Mean is greater than threshold then
                return Arm
            end if
        end if
    end for
    return Arm Randomly
end if
Stop
```

Get Reward Function

```
Start
Increase Counts of arm by 1
if Reward is 1 then
    Increase Values of arm by 1
end if
Stop
```

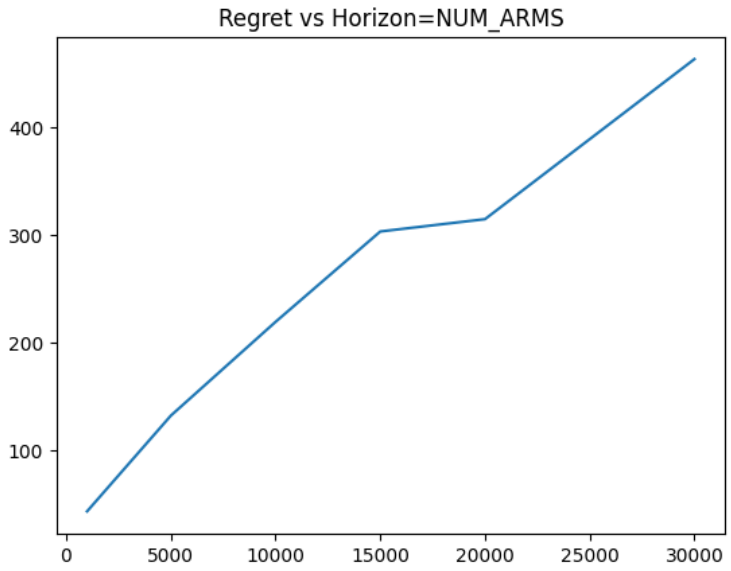

Figure 6: Regret vs Horizon for Task 3

From Figure 6 we see that as number of horizons = number of arms increases, the regret increases. As in this case the C1 condition of GLIE i.e Infinite exploration fails we don't expect the regret to be sub-linear in t which can also be seen from Figure 6.