CS 747: Programming Assignment 1

Dipesh Tamboli - 170070023

September 2020

1 Task T1

- I have used inital empirical probability as zero for all the cases.
- For all sorts of tie-breaking, I have preferred value with the minimum argument(that's what **np.argmax** returns).

1.1 Epsilon-Greedy(ϵG3)

Here, I have considered all initial empirical means to be zero and I let algorithm to pull them on their own instead of doing any Round-Robin pulling.

I am using np.random.binomial function for pulling the arm. I have seeded np.random.seed(randomSeed) in the starting of the code. For explore vs exploit, I am again using using a binomial function with ϵ for getting the flag explore and exploit.
If I get explore, then I am using random integer generator for the getting pulling arm id, or else, pulling the arm having maximum empirical probability.

1.2 UCB

As log(0) is not define, I am using my horizon step iterator from 1 and not zero. As dividing by zero(when pull count was zero) was required, I have put a small value in the denominator which gives a large value and thus that arm gets pulled in the next iteration(same was as what (Round-Robin algorithm would have done).

1.3 KL-UCB

In KL-UCB implementation, I used C=0 as it was mentioned that it yields good result. Also, the term $log(log(t))$ was causing the ror for $t < e$. Also, for finding the max value of the $\mathbf{q}$, I have used a vector ranging from $\hat{p}_a^t$ to 1 in 1000 equal steps to calculate the KL-Divergence vectorically. To be frank, optimising it was quite painful for me and it is still not in the good condition as it takes nearly 12 minutes for horizon of 102400 steps on instance-3.

1.4 Thompson-Sampling

I am using np.random.beta in vector format for getting the beta values at once. Here, I am also calculating the empirical probabilities although it is not required as I used it for checking the output and correctness of the algorithm.

2 Task T2-Thompson-Sampling-With-Hint

Here we are passing/using an additional argument of shuffled true means as hint for some better performance.
I started with using the max(shuffled-true-mean) as the hint and use it only for checking if the empirical probability of the predicted arm provided by the Thompson-Sampling is greater than the 0.95*max-prob or not. If it is less than 0.95*max-prob, then pull the arm with maximum empirical mean. But this results in sort of epsilon-greedy method as the optimal arm whose initial rewards were zero were not getting pulled.

Then I use all the given means in expectation maximization fashion. As we had given initial probabilities as priors.

So I am using a 2d NxN array where is N is the number of arms. And updating it whenver I am getting either a potive reward or negative.

```
for t in range(1, horizon+1):
    arm_id = np.argmax(posterior[:,-1]) #Here, columns of the
    posterior matrix represents the true means in sorted order.

    reward = np.random.binomial(1, arm_prob[arm_id])
    if reward>0:
        posterior[arm_id] *= sorted_p
    else:
        posterior[arm_id] *= sorted_1_p
    posterior[arm_id] /= np.sum(posterior[arm_id])
```

This posterior is representing Bernoulli distribution $B = p^{success}(1-p)^{failures}$. After normalizing the values corresponding to each arm after pulling of that arm, the posterior matrix's row-wise sum equals to one.

Also, I tried to implement this directly by calculating $B = p^{success}(1-p)^{failures}$, but it becomes $(value-less-than-1)^{a-large-number}$ which tends to zero(after loosing computer's precision), and thus divide by zero error was there roughly after 2000 pulls. Thus I opted for the latter approach of multiplying only by the arm got pulled and normalising it thereafter.

3 Task T3

Yes, it is possible to find $\epsilon_1 < \epsilon_2 < \epsilon_3$ and regret of ϵ_2 should be less than the other two.
I observed the trend by changing the ϵ value in steps of the different step size

and found out that it decreases somewhere up to 0.15 and then it increases again.

I have found same epsilon values for all the three instance, $\epsilon_1 = \mathbf{0.002}, \epsilon_1 = \mathbf{0.01}, \epsilon_1 = \mathbf{0.4}$

3.1 Instance 1

- i-1 epsilon:0.002 regret:538.18
- i-1 epsilon:0.01 regret:302.3
- i-1 epsilon:0.4 regret:8200.14

3.2 Instance 2

- i-2 epsilon:0.002 regret:2553.4
- i-2 epsilon:0.01 regret:933.04
- i-2 epsilon:0.4 regret:8225.5

3.3 Instance 3

- i-3 epsilon:0.002 regret:3604.76
- i-3 epsilon:0.01 regret:1204.08
- i-3 epsilon:0.4 regret:16962.94

4 Task T4

4.1 Plots of T1 tasks

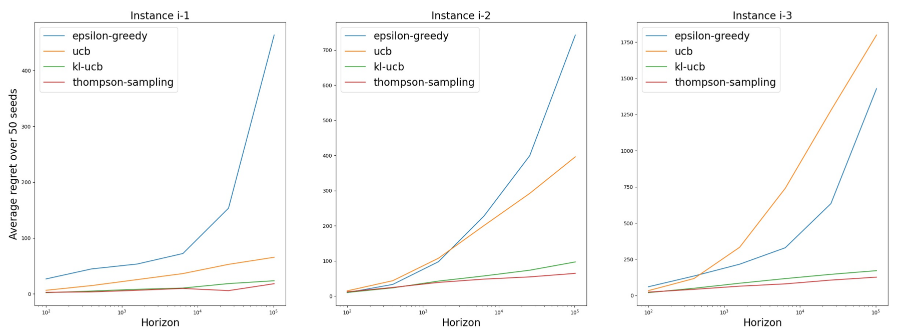

Figure 1: T1 plots

4.2 Observations

- Thompson-Sampling performs better for instance-1,2 and 3.
- Epsilon-Greedy clearly has high regrets for instance 1 and 2, but for instance-3 UCB is performing worst than Epsilon-greedy. But also note the point that if we extrapolate the graph a little further, then UCB's regret is going to be lower than Epsilon-Greedy's regret.
- Thompson-sampling and KL-UCB are showing sub-linear regret for all the times.(UCB's regret is also sub-linear just with the higher slope.)

4.3 Plots of T2 tasks

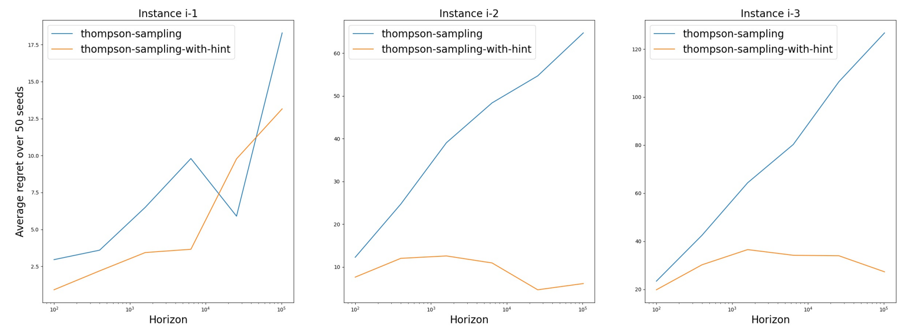

Figure 2: T2 plots

4.4 Observations

- Thompson-sampling-with-hint performs better than without hint for all the instances-1,2,3 and for the horizon 102400.
- For instance-1 horizon 25600, Thompson-sampling-with-hint's regret is higher than the without hint case.
- As Thompson-sampling's regret is sub-linear in nature, Thompson-sampling-with-hint is also sub-linear just the slope being less than the original one.