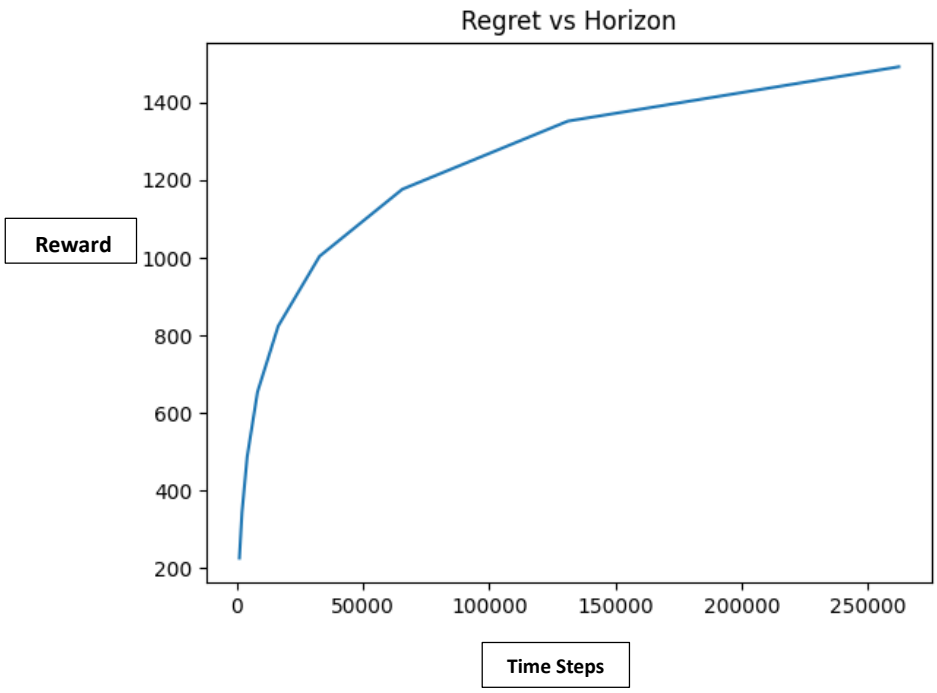
CS 747 Intelligent Learning Agent

Coding Assignment – 1

Task – 1

UCB Plot

The code for implementing UCB algorithm for multi-armed bandits, give pull() method involved returning each of the arms until each one is pulled at least once, and then choosing the arm based on maximum value of $ucb = \hat{p} + \sqrt{\frac{2 * ln(t)}{u}}$

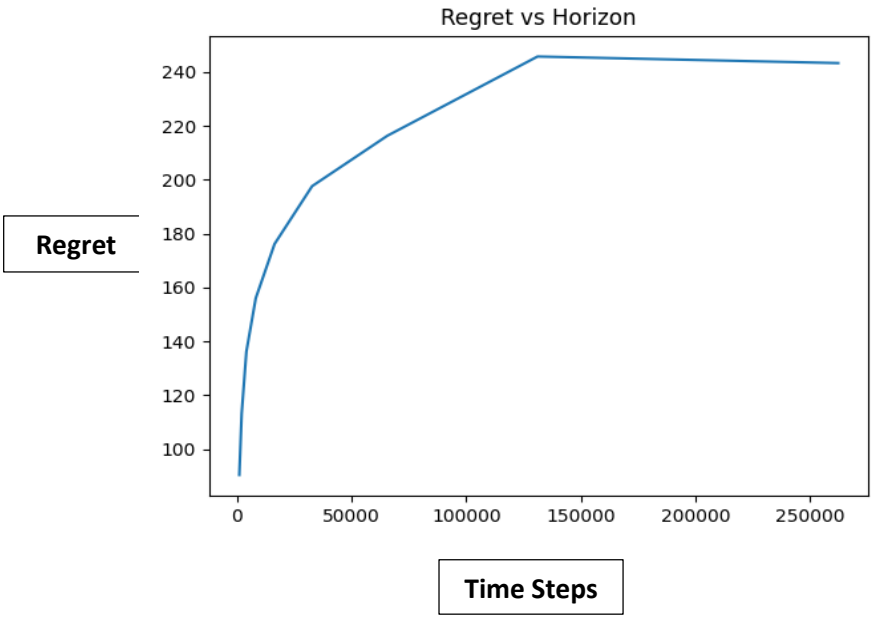


Which is calculated for each arm. Also to make the implementation faster, the calculation of ucb values is vectorized using numpy. For updating the values of p, get reward() method is used, wherein the empirical mean values are updated based on the reward received.

KL_UCB Plot

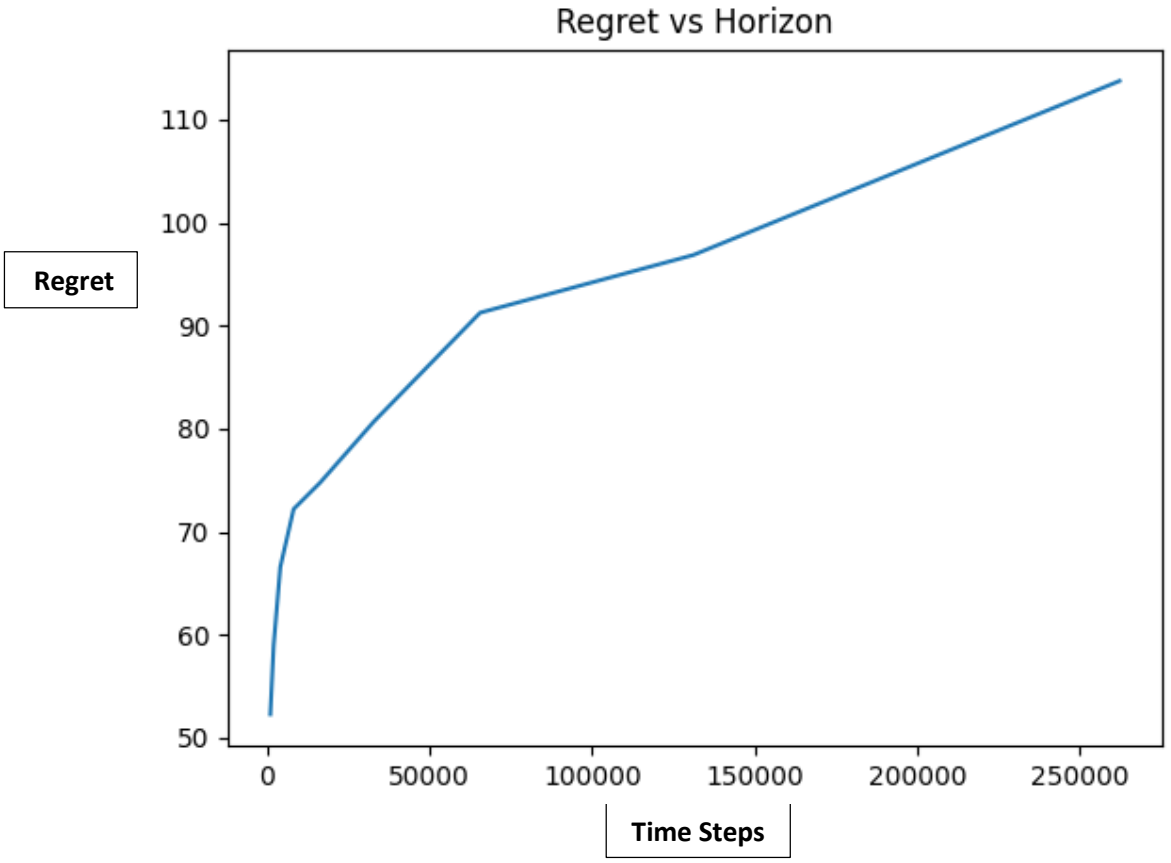
In order to implement KL-UCB algorithm, firstly a function is defined to calcu□late KL divergence. Also a function to implement bisection method(binary search method) for any given function is used. The get pull() method involves calculating the ”q” value such that KL(p, q) = ln(t) + cln(ln(t)) / u

The arm with maximum above value is pulled. Again, get reward() is only used to update the empirical mean values using the reward



Thompson Sampling Plot

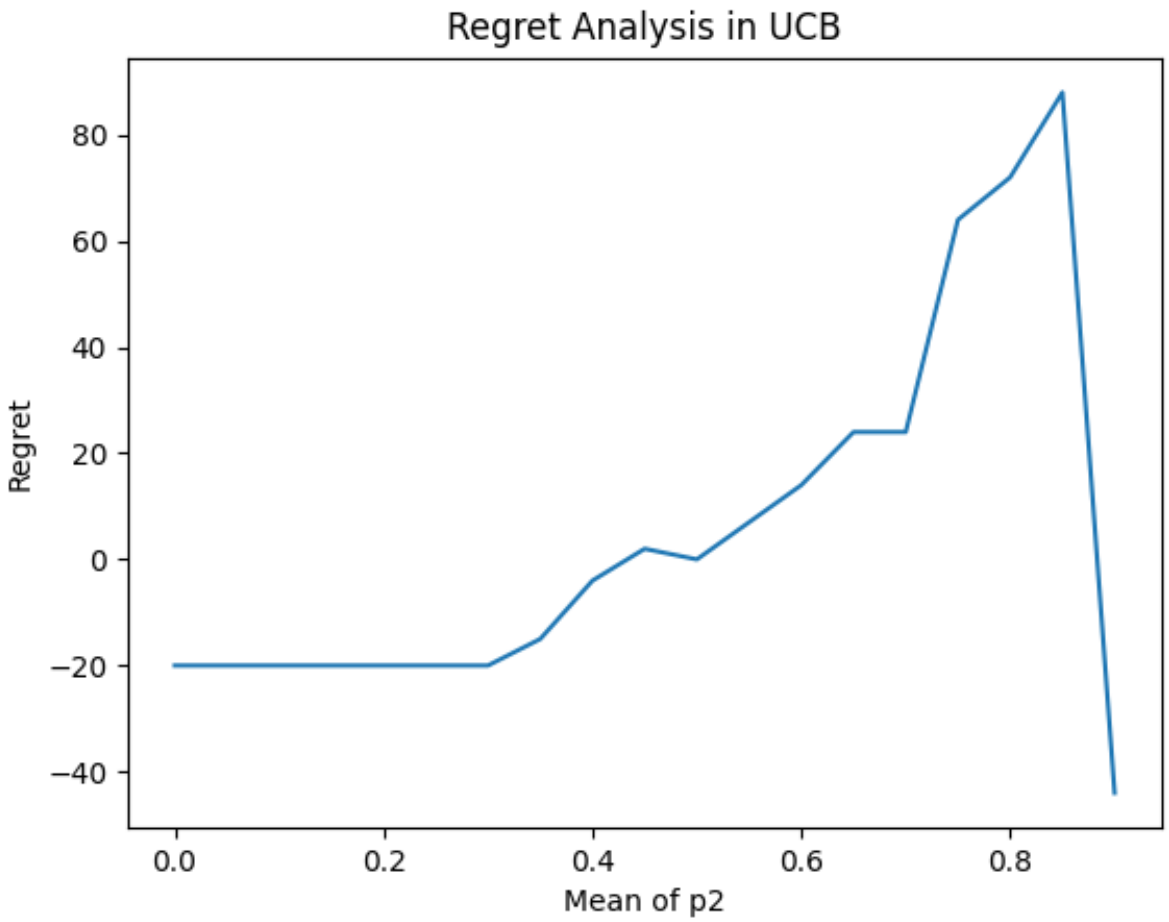
This is implemented by sampling from beta distribution defined by parameters α = #successes and β = #failures for each arm, and then pulling the arm having maximum sample value. Here the



get_reward function is used for updating the number of failures and successes by pulling a particular arm. x ~ β(1 + #successes, 1 + #failures)

Task-2

Part-A



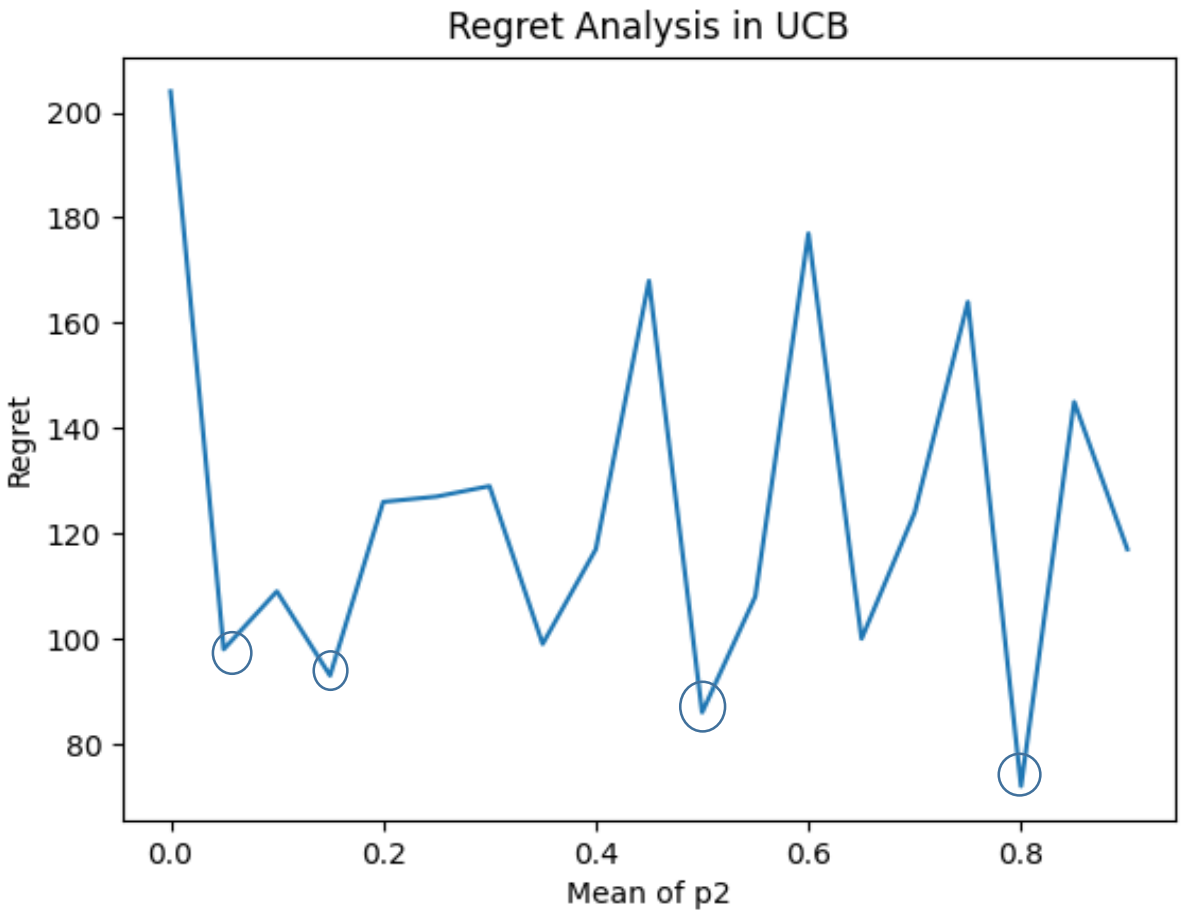
As the mean of p2 increases from p2, the regret increases, and at 0.8 it becomes maximum

We can observe that there is a significant amount of dependence of regret on the mean of bandit instance, when the mean of both instances become same, both are equally desirable hence in that case the regret decreases

Part-B (UCB)

From the graph we can observe that the lower of the peaks are decreasing, and it shows that the overall trend is decreasing.

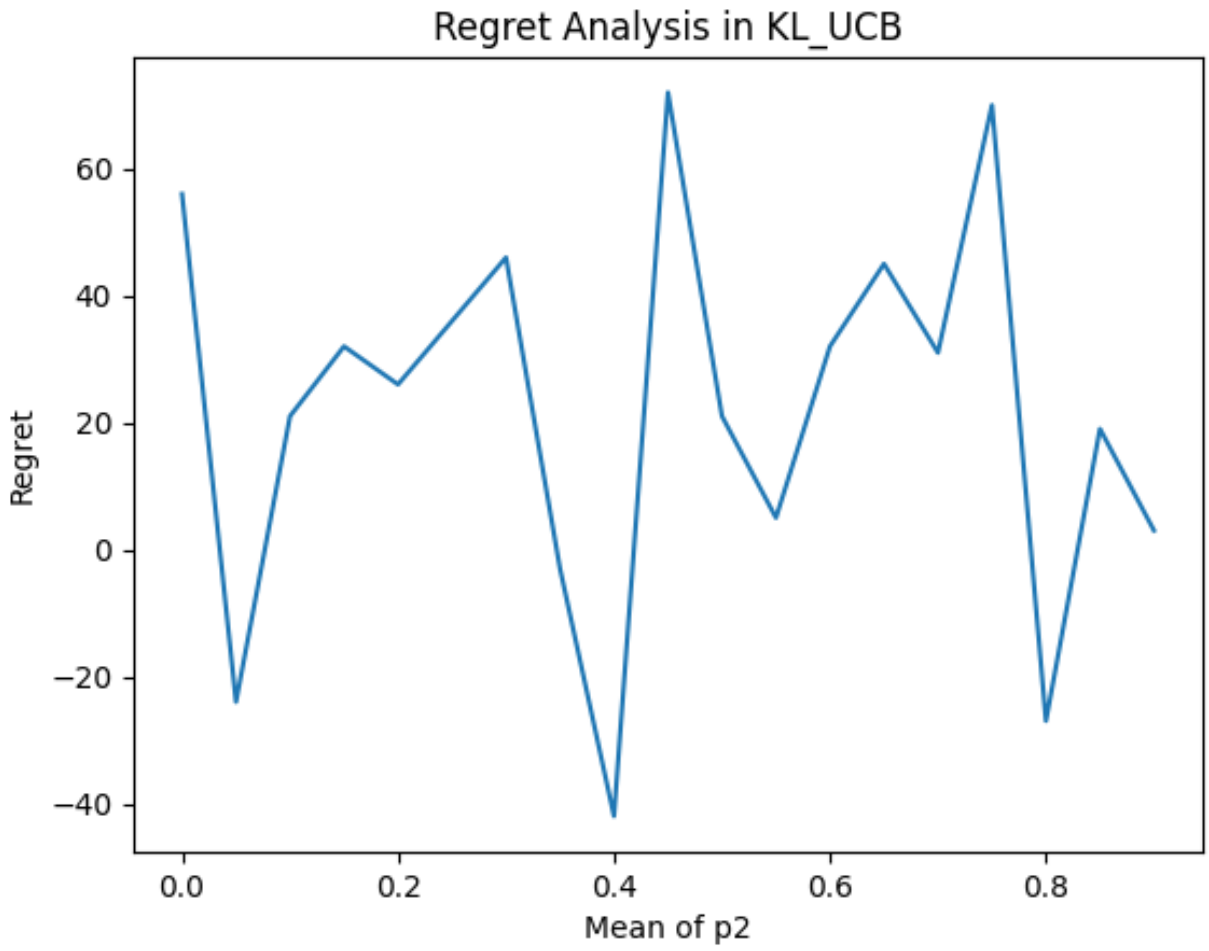
In first step, $mean_{p2} = 0$ and $mean_{p1} = 0.1$ – in the case the three is very high chances that bandit is optimal acc. To UCB, and the logic of model will select it the most, just in case of the exploration the model choose the p2 otherwise it



will choose the p1.

So extending the same idea we can tell that as the p1, p2 both increases, but there is high chances that the p2, in exploration phase get good reward and p1 get bad reward, and p2 outperform in exploitation.

Part-B (KL_UCB)



Keeping the situation the same, using we are KL_UCB algorithm, we are tightening the upper bound, so out of 19 instances it gives less lesser regrets. And the upper bound tightening to increase chances of getting more optimal exploitation, there are several instances of consequent optimal exploitation.

How I accomplished task-2?

There were just 2 lines to change from my understanding,

For part-A, I changed task2p1s and made a list of length=19, and containing all values = 0.9, and run a comprehensive for loop to create values for task2p2s, which have values from 0 – 0.9 increment of 0.05 each.

For part-B, here is the code snippet which I changed

```
task2p2s = [round(i/100,2) for i in range(0,95,5)]
task2p1s = [round(i + 10e-2,2) for i in task2p2s]
```

We can see that task2p2s is 0 to 0.9 with increment of 0.05 and using that I created task2p1s, which essentially have values with increment of 0.1, so the difference remains 0.1

How did i task-3?

In task-4 I applied KL_UCB algorithm, by just changing its mean value with **expected mean**, which takes care of faulty instances

How did I task-4?

I simply applied Thompson sampling algorithm, for 2 bandits instances, I changes the code and the array dimension for that, I the get_pull I created an random bandit instance generator to imitate the process of set_pulled.

Else I just changes the list to arrays so it can save and modify values accordingly

References

Batched Thompson Sampling: Link

Thompson Sampling: An Asymptotically Optimal Finite Time Analysis: Link