CS747-Assignmenent 1

Palle Bhavana - 210050111

September 2023

Contents

1 Task 1

Task is to generate regrets for different horizons using the algorithms **UCB**, **KL-UCB** and **Thompson Sampling**.

1.1 UCB

Upper-Confidence Bound arm selection algorithm uses uncertainty in the action-value estimates for balancing exploration and exploitation. The arm with highest ucb value is choosen to pull next, which contains explore and exploit terms in the ucb value expression.

$$ucb_a^t = \underbrace{\hat{p}_a^t}_{\text{Exploit}} + \underbrace{\sqrt{\frac{2\log t}{u_a^t}}}_{\text{Explore}}$$

1.1.1 Implementation

`give_pull:`
The strategy used to pull the arm is: Explored each arm at the start for each horizon, so that explore term is non-zero for ucb value to get defined. For the next pulls, compute the ucb value for every arm and get the index of the maximum ucb value arm.
`get_reward:`
Incremented the number of times the arm with the index is pulled and updated the value of the expected probability of the arm.

$$\hat{p}_a^t = (\frac{count-1}{count}) * \hat{p}_a^{t-1} + \frac{1}{count} * reward$$

1.1.2 Plots and Observations

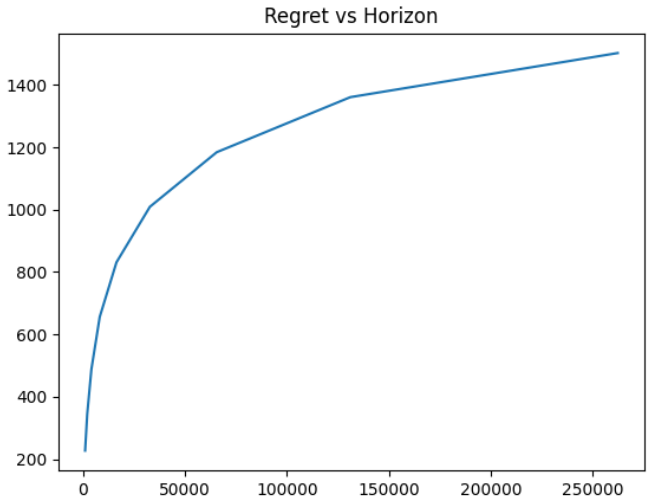


Figure 1: Regrets vs Horizon plot for UCB algorithm.

The regret with horizon is not increasing linearly with the time instances, which matches with the expected regret for UCB algorithm

$$E[Regret] = O(K \log T)$$

1.2 KL-UCB

Kullback-Leibler-UCB algorithm is identical to UCB algorithm, except for a different definition of the upper confidence bound and it is more optimised over the UCB algorithm

$$ucb - kl_a^t = max\{q \in [\ \hat{p}_a^T, 1] \texttt{ s.t } u_a^T KL(\hat{p}_a^T, q) \leq \ln t + c \ln \ln t \texttt{ where c} \geq 3\}$$

To make computation easy and for the pratical purposes, c is substituted as 0

1.2.1 Implementation

`give_pull:`
The strategy used to pull the arm is: Explored each arm at the start for each horizon, so that explore term is non-zero for kl-ucb value to get defined. For the next pulls, compute the kl-ucb value for every arm and get the index of the maximum kl-ucb value arm.
To compute the kl-ucb value, function ucb_value is defined
`ucb_value:`
Used Iterative binary search for computing the greatest kl-ucb value that is less than or equal to k and kept the precision as 0.01.

`get_reward:`
Incremented the number of times the arm with the index is pulled and updated the value of the expected probability of the arm.

$$\hat{p}_a^t = (\frac{count - 1}{count}) * \hat{p}_a^{t-1} + \frac{1}{count} * reward$$

1.2.2 Hyper parameters

1. c = 0
As suggested, for making the computation fast.
2. precision = 0.01
I have computed the regret values for the precision 0.001, the values are almost same as with precision 0.01 but it is also taking relatively higher time compared to this.

1.2.3 Plots and Observation

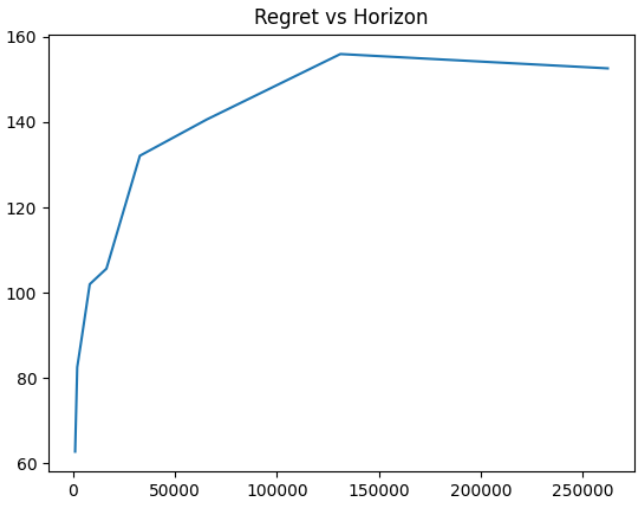


Figure 2: Regrets vs Horizon plot for KL-UCB algorithm.

1. The regret with horizon is not increasing linearly with the time instances, which matches with the expected regret for UCB algorithm

$$E[Regret] = O(K \log T)$$

2. Regrets of KL-UCB is comparitively less as compared to UCB, it is more optimized

1.3 Thompson Sampling

Thompson sampling algorithm balance between exploiting what is known to maximize immediate performance and investing to accumulate new information that may improve future performance. It considers the beta prior on the probability of success for each arm and after every time instance it pulled, it updates the distribution of the probability of success.

$$x_a^t = Beta(s_a + 1, f_a + 1)$$

1.3.1 Implementation

`give_pull:`
Computed the predicted values from the beta distribution and returns the arm which have the maximum predicted value. This algorithm does not need every arm to be pulled atleast once, because initially the distribution will be uniform.
`get_reward:`
Based on the reward, the success count or the failure count will be incremented.

1.3.2 Plots and Observation

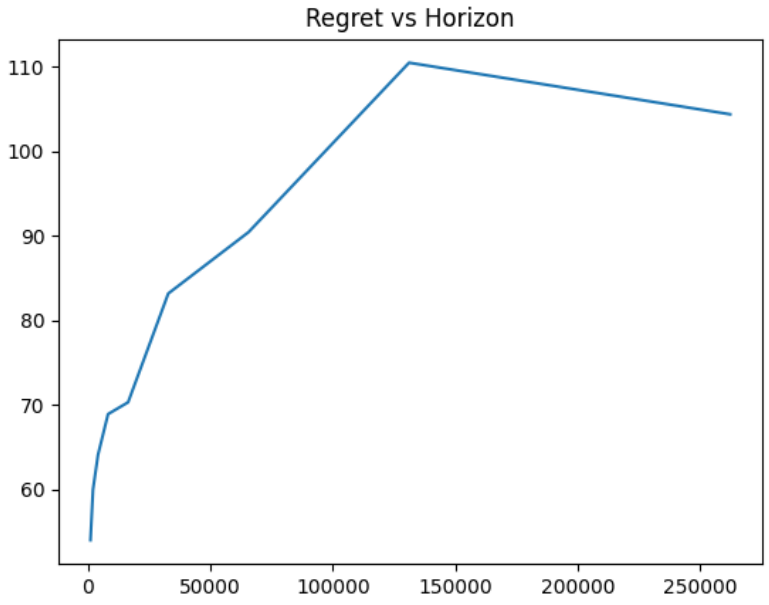


Figure 3: Regrets vs Horizon plot for Thompson Sampling algorithm.

1. The regret with horizon is not increasing linearly with the time instances,

which matches with the expected regret for UCB algorithm

$$E[Regret] = O(K \log T)$$

2. Regrets of Thompson Sampling is less than KL-UCB and UCB.

2 Task 2

The task is to generate the plots on different conditions of probabilities of two arms and give explanations to the plots.

2.1 Part A

Plotted the graph for UCB algorithm with varying p2 values and gave the explanation to the observed graph.

2.1.1 Plots and Observations

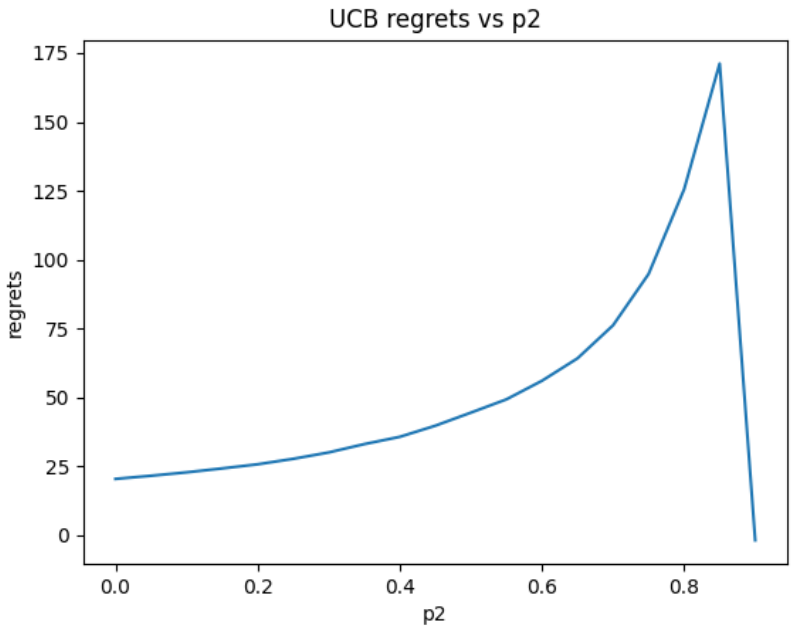
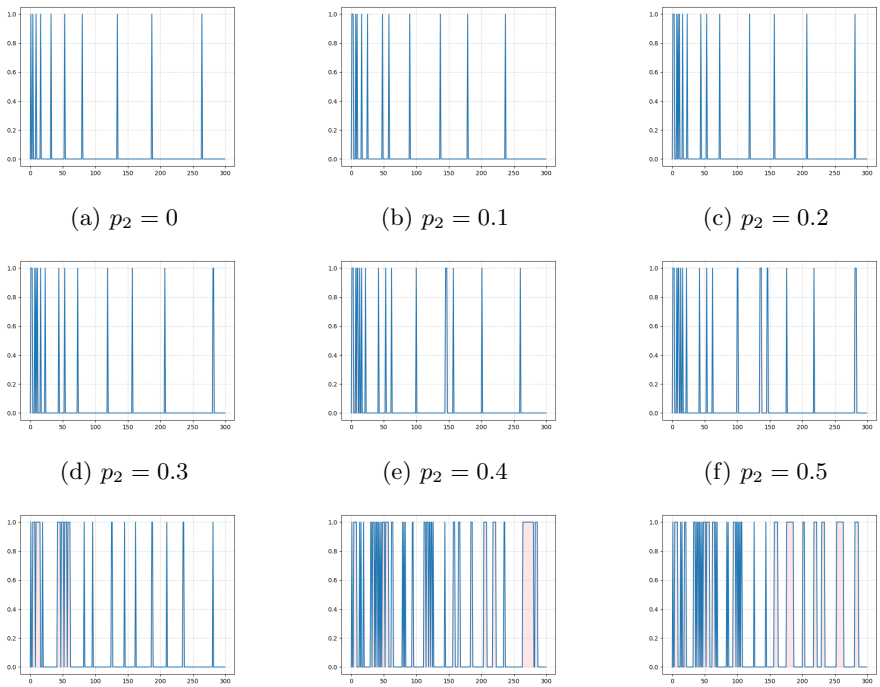


Figure 4: Regrets vs p2 plot for UCB algorithm.

1. Regrets increase from 0 to 0.8 and at 0.9 becomes 0. The regret has to be 0 at 0.9 becuse both the arms will have same proabability, so no regret.
2. Summary is that as difference between 0.9 probability and other arm decreases, regret increases.

2.1.2 Explanation and Plots



(a) $p_2 = 0$ (b) $p_2 = 0.1$ (c) $p_2 = 0.2$

(d) $p_2 = 0.3$ (e) $p_2 = 0.4$ (f) $p_2 = 0.5$

(g) $p_2 = 0.6$ (h) $p_2 = 0.7$ (i) $p_2 = 0,8$

Figure 5: UCB: Rate of choosing best arm from 300 stimulations with increasing p2

For understanding the Figure: 4 plot, I have plotted the graph on Rate of choosing best arm from 300 simulations for all the p_2 values, the observation of these plots are giving Explanation of why the regret is increasing with increasing the value of p_2. The code for this plotting is written in `for_plot` function in task2 file.

The UCB algorithm have extermely high fluctuations in its rate of choosing the best arm in the initial time, because it is in the exploring phase and the UCB value of all the arms are high for all values of p_2.
Analysed it in two parts:
When the difference between the best arm and the other arm is large, As the time progresses, the UCB components becomes much smaller for all arms, and the UCB function representation of each arm converges towards the average reward mean of each arm. Thus, the arm with the higher mean becomes more distinguishable by the algorithm and becomes more frequently picked as the trial progresses. This can be seen in first three figures, when the values of p_2

are lower, p2 exploration happens but relatively for less time, most of the time best arm is choosen by the UCB.
When the difference between the best arm and the other arm is small, In the previous case, when the difference is large, it took less number of times for the UCB algorithm to explore. Now since the difference is small it takes higher number of times to explore than the previous cases, which generates more regret as observed in the last 3 plots for p2 = 0.6, 0.7 and 0.8.
When the difference between the two arms is zero
Choosing any arm in those does not cause any regret, so there is the drastic decrease from 0.8 to 0.9.

2.2 Part B

Plotted the graphs for UCB and KL-UCB for different values of p_2 and gave the explanation to the difference.

2.2.1 Plots and Observations

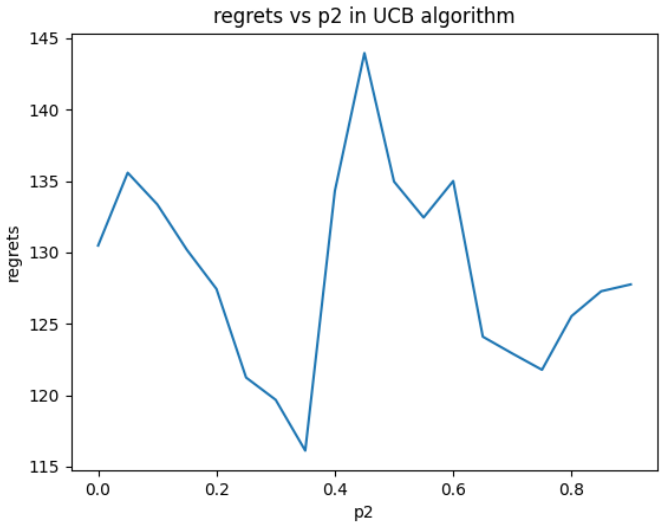


Figure 6: Regrets vs p2 plot for UCB algorithm.

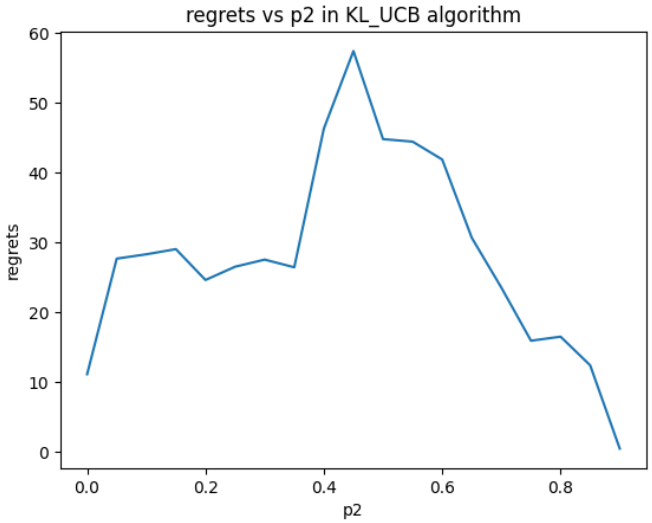
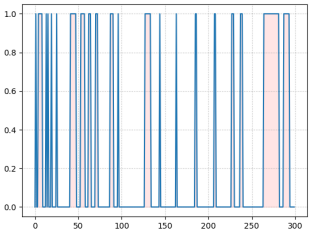


Figure 7: Regrets vs p2 plot for KL-UCB algorithm.

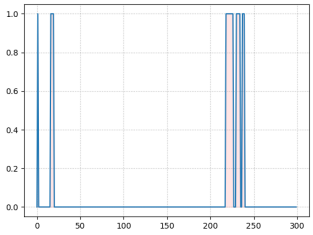
1. The Regret value is maximum at 0.5 value of p2, for both the algorithms.
2. The Regrets of KL-UCB algorithm are less than UCB algorithm.

2.2.2 Explanantion

1. Maximum regret at 0.5 Probability
Variance: Since it is the beneroulli bandits the variance of regrets is maximum at 0.5 probability, which means it takes more explorations for the bandit to get near to its arm mean, compared to other values, which intern increases the regrets.
2. Difference in regret values for KL-UCB and UCB:
Explorations:



(a) Plot of UCB algorithm with $p_2 = 0.75$



(b) Plot of KL-UCB algorithm with $p_2 = 0.75$

The number of explorations of KL-UCB algorithm is much much less than UCB algorithm, which can be observed from the graph plot of arm pulled with the time instances, which intern decreases the regret by a great amount

3 Task 3

Used Thompson sampling with a variation to it. The probability of it is real success given reward value is computed by the formula.

$$P(success|reward) = \frac{p(reward\ \&\ success)}{p(reward)}$$

$$p(reward\ \&\ success) = \hat{p} \cdot ((1\text{-}self.fault) \cdot reward + 0.5 \cdot self.fault)$$

$$p(reward\ \&\ failure) = (1 - \hat{p}) \cdot ((1\text{-}self.fault) \cdot (1 - reward) + 0.5 \cdot self.fault)$$

$$p(reward) = p(reward\ \&\ success) + probability(reward\ \&\ failure)$$

here $\hat{p}$ is the expected reward

3.1 Implementation

`give_pull:`
It is same as in normal thompson sampling, predicts the value from the beta distribution and return the index of the maximum value.
`get_reward:`
The value of the success of the arm is increased by P(success—reward), computed by the equation given above and the failure is increased by 1-P(success—reward).

3.2 Plots and Observations

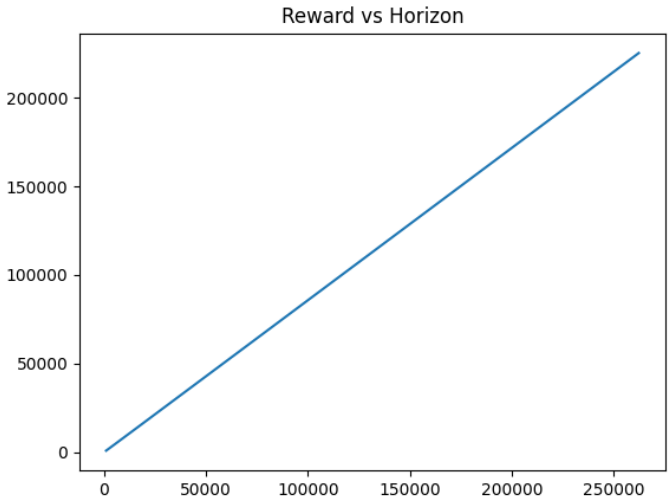


Figure 9: Rewards vs Horizon plot for Task 3.

1. The Reward is increasing linearly with time

4 Task 4

Used Thompson Sampling with a slight modification to it, as there are two bandits, stores two success, failure and the predict values arrays for each bandit. But when you are predicting the arm, take the mean of two bandits and return the index that have maximum value, so that reward gets maximized. I have tried taking min, max of two bandits and return the index but these does not perform better than mean value.

4.1 Implementation

`give_pull:`
Returns the index with maximum mean value of the predict values of two arms.
`get_reward:`
Based on the set-pulled argument only the corresponding bandits success and failure arrays are updated, same as in Thompson Sampling.

4.2 Plots and Observations

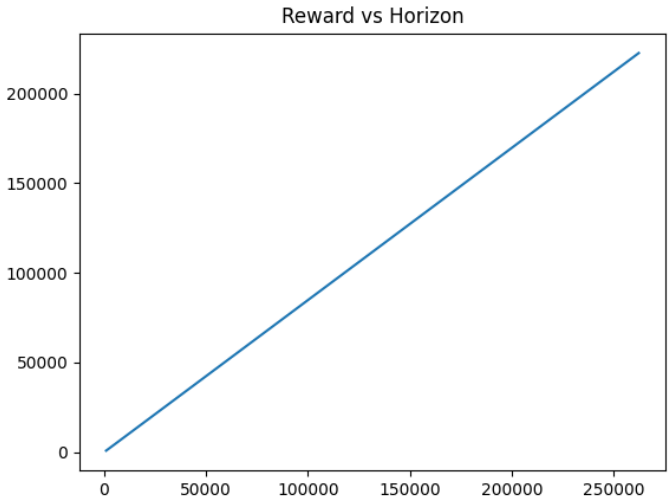


Figure 10: Rewards vs Horizon plot for Task 4.

The Reward is increasing linearly with time.

5 References

Resources used to generate plots and reports for this assignment apart from the lecture slides.

5.1 UCB

https://www.geeksforgeeks.org/upper-confidence-bound-algorithm-in-reinforcement-learning/
https://bochang.me/blog/posts/bandits/
https://kfoofw.github.io/bandit-theory-ucb-analysis/

5.2 Thompson Sampling

https://web.stanford.edu/~bvr/pubs/TS_Tutorial.pdf

5.3 Graphs Plotting

https://learnpython.com/blog/average-in-matplotlib/