CS 747 : Assignment 1

Shalabh Gupta (180050095)

22 September 2020

1 T1

Method and Implementation of epsilon-greedy, ucb, kl-ucb and thompson-sampling.

1.1 Epsilon-Greedy Implementation

- First sample all arms once in a round robin fashion using a for loop. **Empirical mean, Total reward** and number of pulls for each arm updated in the respective numpy arrays.
- Then epsilon-greedy3 followed till horizon, picking the arm with **maximum empirical mean** till now with probability epsilon and exploring with probability 1-epsilon.
- **Argmax** function of numpy was used to pick arm with maximum empirical mean from array mean_reward which picks up the smallest indexed arm to resolve ties.

1.2 UCB

- First sample all arms once in a round robin fashion using a for loop. Empirical mean, Total reward and number of pulls for each arm updated in numpy arrays.
- Then UCB algorithm used to sample arms till horizon, picking the arm with the **maximum upper confidence bound** found using the UCB expression (stored in *ucb_array* in the code).

$$ucb_a^t = p_a^t + \sqrt{\frac{2ln(t)}{u_a^t}}$$

- Argmax function of numpy used to pick max UCB arm, resolving ties with smallest indexed arm.

1.3 KL-UCB

- Same round robin fashion sampling initially as above algorithms
- KL-UCB algorithm used to sample arms till horizon, picking the arm with maximum value of q obtained by finding an approximate solution of the following expression using **binary search**.

$$kl - ucb_a^t = \max_q \{u_a^t * KL(p_a^t, q) \leq \ln t + c * \ln(\ln(t)) \} \text{ while } q \in [p_a^t, 1]$$

- Precision in binary search set to 1e-4 and c set to 3 after experimenting.
- To pick arm with maximum value of q, Argmax function of numpy used which picks up smallest indexed arm from klucb_array in case of ties.

1.4 Thompson-Sampling

- No round robin sampling in this case.
- **Sucesses** and **failure** arrays additionally present now to keep track of 1s and 0s encountered for each arm pulled. Being updated according to every new reward encountered.
- Maintaining **beta distributions** for each arm using parameters success[arm]+1, failure[arm] + 1, and sampling from these distributions using numpy.random.beta
- Arm having max value on sampling is chosen by applying Argmax function in numpy on thomp_arr which has sampled numbers. Ties again resolved by choosing smallest indexed arm.

2 T2 : Thompson Sampling With Hint

2.1 Method

Since we have this additional knowledge of sorted true means, we can form **likelihood distributions** for these mentioned probabilities against all arms. In other words, I keep a likelihood matrix, say M, whose i,j entry denotes the probability that jth probability corresponds to arm i in our bandit instance. jth probability is the jth smallest probability(since we are assuming sorted true means). This will be purely updated based on the data we will receive along with the sorted true means, and since we know that the last column of this matrix corresponds to probabilities of arms having largest true mean p* (because of our defined convention), we can find on every step of our algorithm the arm having the maximum probability of having highest true mean and pull that arm. Then based on the reward , we can update the probability distributions by multiplying with p if 1 comes, or 1-p if 0 comes.

2.2 Implementation

- The 2d numpy array likelihood_mat is exactly matrix M described in the method subsection, initialised to all 1. Let the sorted true means be [p1,p2,p3,...,p*] given by numpy array hints passed as argument to the function.
- The j^{th} column denotes the bernoulli kind likelihood probability distribution for probability p_j for each arm. So i,j element of likelihood_mat is roughly the likelihood of arm i having true mean p_j, calculated on basis of observed data till now.

$$likelihood\_mat[i][j] \propto (p[j])^{succ[i]}(1-p[j])^{fail[i]}$$

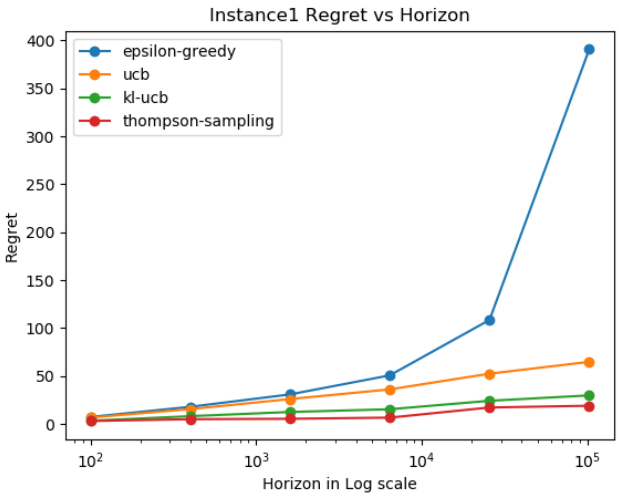
- Now we find arm having highest likelihood for last column of likelihood_mat i.e. corresponding to p* using argmax function of numpy (resolving ties by smallest indexed arm again) and pull this.
- Now if reward obtained is 1, we update our likelihood for this arm and probabilities [p1,p2,p3...p*] by multiplying with the corresponding prob. While if reward is 0, we update by multiplying with 1-prob. Note that we are only updating the likelihood of the arm we picked in a step.
- The likelihood_mat is being normalized row-wise to keep the sum of rows = 1 (rel. probabilities) so that all arms have same scale wrt the maximum true mean(last column).

3 T3

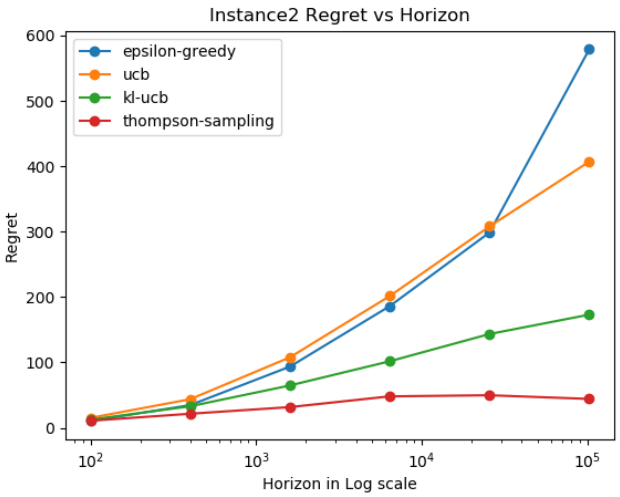
- Instance 1 : Epsilon_1 = 0.001 , Epsilon_2 = 0.003, Epsilon_3 = 0.03
- Instance 2 : Epsilon_1 = 0.001 , Epsilon_2 = 0.01, Epsilon_3 = 0.05
- Instance 3 : Epsilon_1 = 0.001 , Epsilon_2 = 0.01, Epsilon_3 = 0.05

4 T4

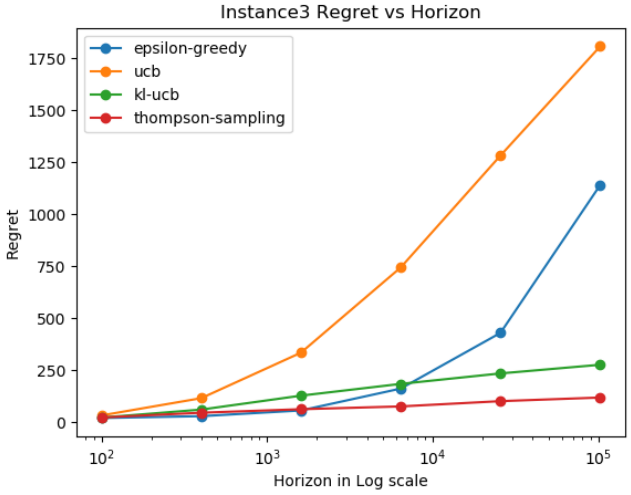
Following are the plots for the performance comparison of algorithms in T1 i.e. Epsilon-Greedy, UCB, KL-UCB and thompson sampling on each of bandit instances 1,2 and 3. Y-axis shows regret and X axis has horizon in log scale.



(a) Instance 1



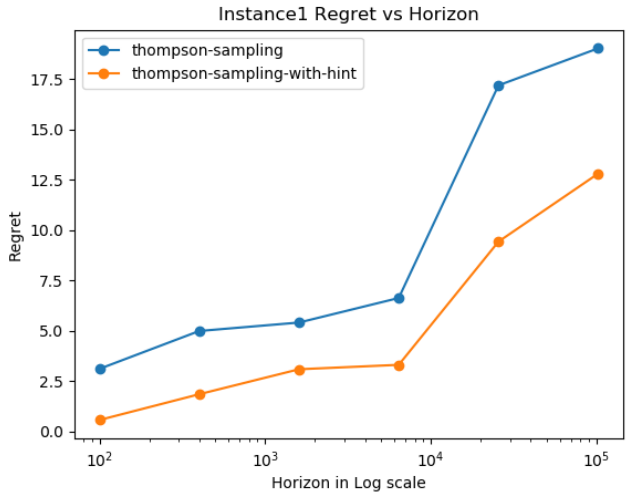
(b) Instance 2



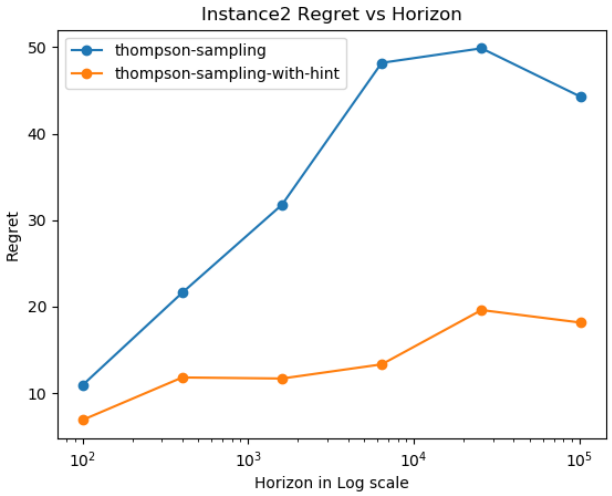
(c) Instance 3

Figure 1: Regret vs Horizon Comparisons of T1 algorithms

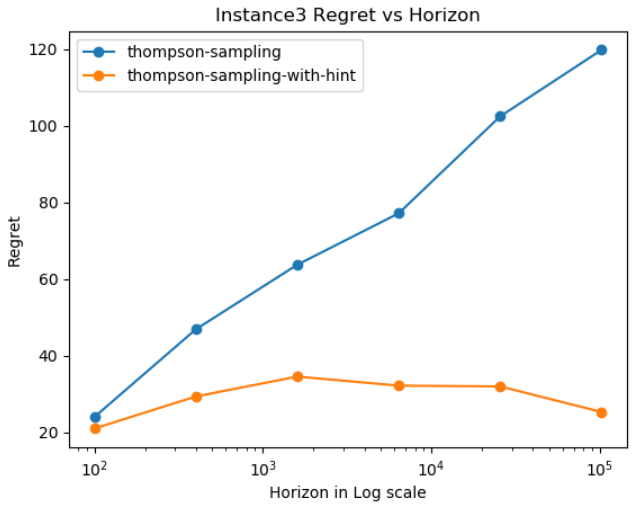
Following are the plots for the performance comparison of thompson-sampling and thompson-sampling-with-hint on each of bandit instances 1,2 and 3. Y-axis shows regret and X axis has horizon in log scale.



(a) Instance 1



(b) Instance 2



(c) Instance 3

Figure 2: Regret vs Horizon Comparisons of Thompson Sampling without and with hint

5 Conclusions/Interpretations

- As expected in T1 algorithms, regret bound of UCB, KL-UCB and thompson sampling is **linear** with log of horizon with thompson sampling having the tightest bound, KL-UCB having the second tightest followed by UCB.
- Epsilon greedy algorithms give linear regret and thus the graph is **exponential** with respect to log horizon. The fact that in instance-3, the regret is higher in UCB than epsilon-greedy is because of the higher constant that accompanies asymptotic bound due to arm probabilities being

very close to each other, while UCB regret being **O(log(T))** and epsilon-greedy regret being **O(T)** as expected and can be seen in the plot.

- This shows that on much higher horizons, the order of regrets would be **thompson-sampling** $<$ **KL-UCB** $<$ **UCB** $<$ **epsilon-greedy**.
- From T2 plots for each instance, we can clearly see how much better our algorithm can work in reducing regret on having even a small amount of extra information using bayesian statistics.
- Regret in thompson-sampling-with-hint decreases a bit for instance 3 on larger horizons. This may be happening due to almost always picking of the optimal arm by the algorithm(which should ideally keep regret constant) and our algorithm getting **lucky** on most later pulls and producing a 1 more than it should(p*) leading to much negative values of p*-1 (this is what is added to the regret in each step)
- It is seen that on increasing epsilon in epsilon-greedy algorithm, when averaged over the 50 seeds on horizon 102400, the regret first decreases and then increases. The minima obtained is illustrative of the balance between **exploitation and exploration** that needs to be maintained to get low regret.
- I observed that thompson-sampling was giving better regret values without round robin while thompson-sampling-with-hint gave lower regret on 1 round robin pulling of arms. This is why I have compared these two optimal performances of the algorithms against each other.