CS747 - Programming Assignment 1

Raaghav Raaj

September 9, 2021

Brief Overview of `bandit.py`

The code comprises of two classes:

- **`arm`** - The class has two attributes: *outcomes* and *probabilities*, and one method: `pull`. An object of the class is initialised using these attributes. The attribute *outcomes* is a list of possible rewards in that arm whereas the attribute *probabilities* is a list of probabilties at which the corresponding reward can be achieved.

  The attributes have been implemented in such manner to adapt to the instances of **Task-3** and **Task-4**.

  The method `pull` does not take any arguments. It generates a random number between 0 and 1 from a uniform distribution, and with the help of the attributes *outcomes* and *probabilities* returns a reward.

- **`BernoulliBandit`** - The class has two attributes: *instance* and *arms*, and one method: `runAlgo`. An object of this class, is initialised using these attributes. *instance* is the path of the instance provided while initialising an object and *arms* is a list of objects of class `arm` which are generated using the instance. Please check the code for more information.

  `runAlgo` method takes the remaining command line arguments: *algorithm, randomSeed, epsilon, scale, threshold, horizon.* The *numpy* random number generator is seeded using the *randomSeed* value provided to make the output deterministic. Next, an appropriate sampling algorithm is run after checking the *algorithm* provided via command line.

Further, the code has 4 sampling algorithms which will be described in the next section.

Task 1 - Sampling Algorithms Implementation

- **Epsilon Greedy Algorithm** - The algorithm implemented is the 3rd flavour of this category of sampling algorithm. In this method, at every time t, we generate a random number μ from Unif $(0, 1)$
  - if $\mu < \epsilon$, pull an arm from the list of arms randomly,
  - else, pull the arm with maximum empirical probability at time t
  - `np.argmax()` is used for the above task, which resolves ties by returning the lower index
- **Upper Confidence Bound Algorithm** - Rather than exploring with constant probablity by randomly choosing an arm, the UCB algorithm changes its exploration-exploitation balance as it gathers more knowledge of the bandit instance. Each arm is associated with an **UCB** value which is determined as follows:

  $$ucb_a^t = \widehat{p_a^t} + \sqrt{\frac{c \ln t}{u_a^t}}$$

  where ucb_a^t is the UCB value of the arm a at time t, $\widehat{p_a^t}$ is the empirical probability of arm a at time t, u_a^t is the number of pulls of arm a until time t, and c is the confidence value, also known as *scale*, which is altered to get the optimal result. For **Task-1**, the default value is set as 2. For **Task-2**, this value will be varied to determine the optimal *scale*.
  - initially all the arms are pulled once for the algorithm to begin.
  - at time $t \geq N$, pull the arm with maximum value of ucb_a^t
  - `np.argmax()` is used for the above task, which resolves ties by returning the lower index
- **KL-UCB Algorithm** - The idea is similar to the **UCB Algorithm**, but with a different definition of the upper confidence bound. Here, we define the **UCB-KL** value which is determined as follows:

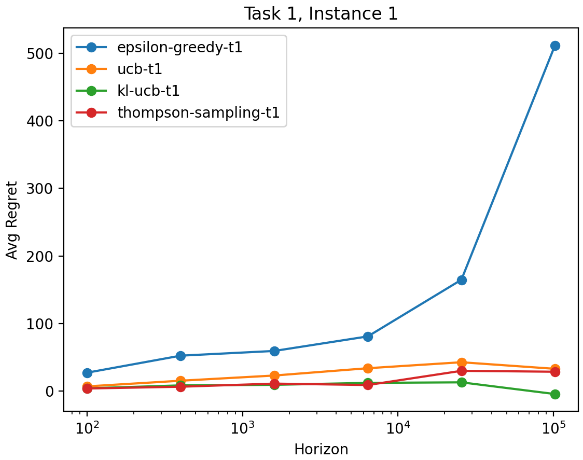
  $$\text{ucb-kl}_a^t = \max\{q \in [\widehat{p_a^t}, 1] \mid KL(\widehat{p_a^t}, q) \leq \frac{\ln t + 3 \ln \ln t}{u_a^t}\ \}$$

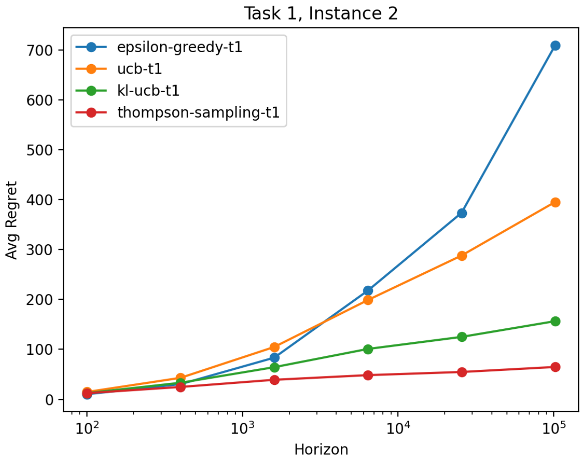
  where ucb-kl_a^t is the UCB-KL value of arm a at time t, the other variables have the same meanings as defined in the UCB algorithm and the kl-divergence of two bernoulli variables, $KL(x, y)$, is defined as

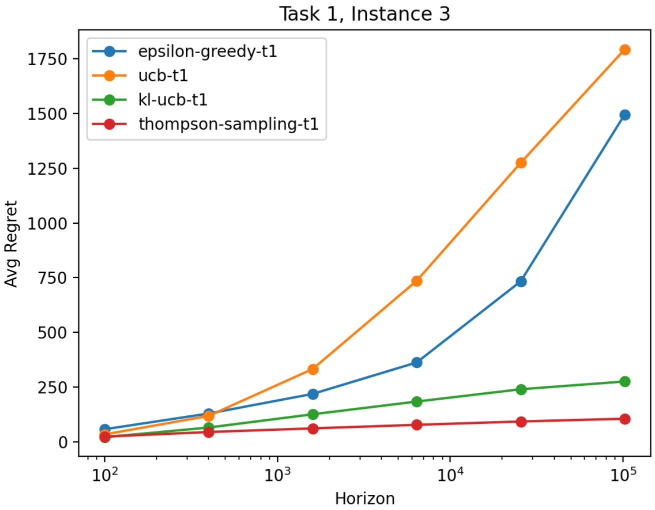
  $$KL(x, y) = x \ln \frac{x}{y} + (1 - x) \ln \frac{1 - x}{1 - y}$$

  - initially all the arms are pulled once for the algorithm to begin.
  - at time $t \geq N$, pull the arm with maximum value of ucb-kl_a^t
  - `np.argmax()` is used for the above task, which resolves ties by returning the lower index

- **Thompson Sampling** - This particular algorithm is fairly different from the other algorithms. We use the *Beta* distribution with the parameters $\alpha = s_a^t$ and $\beta = f_a^t$, where s_a^t is the number of successes of arm a and f_a^t is the number of failures of arm a until time t.
  - for all the arms in the bandit, we sample a value from $Beta(s_a^t + 1, f_a^t + 1)$ which represents a *belief* about the true mean of arm a.
  - pull the arm with the maximum sampled value
  - `np.argmax()` is used for the above task, which resolves ties by returning the lower index





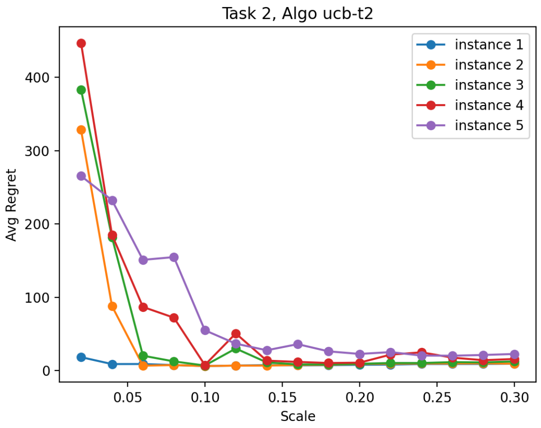


Observations and comments:

- for instance 1 and 2, the results and plots are as expected. Epsilon greedy shows exponential regret w.r.t log scale horizon whereas the other algorithms show linear regret.
- for the instance 3, the results seem slightly surprising as the UCB algorithm gives higher regret than the epsilon greedy algorithm. This is only because the `instance-3` has much larger number of arms than the previous two instances.

  Here, UCB algorithm has not yet explored the arms well which is why the regret is not minimised. For higher horizons of the order $10^7, 10^8$, the UCB is expected to outperform the ϵ-greedy method, which is also suggested from the rapidly increasing and high slope of its curve compared to that of the UCB algorithm.

Task 2 - Scale Optimisation in UCB Algorithm



The optimal scale values for each instance is as follows:

Instance	Mean Rewards	Optimal Scale, c
1	0.7, 0.2	0.04
2	0.7, 0.3	0.06
3	0.7, 0.4	0.10
4	0.7, 0.5	0.10
5	0.7, 0.6	0.20

As it can be seen, with each instance, the mean rewards are getting closer. This leads to confusion in the algorithm while exploiting and thus the exploration has to be increased. Hence, we find an increasing trend in the optimal value of scale, c, with every next instance.

Task 3 - Minimising regret for Bandit of Arms with Multiple Rewards

- **Overview:** The idea behind the algorithm is same as that of the UCB algorithm for a bandit with binary reward arms . As we have seen earlier, the algorithm provides us with a better and more calculated exploration vs exploitation. It is basically the spirit of optimism where we look to exploit the arm with the highest empirical mean reward, and explore the arm we haven't seen much of.

  Slight change of notations as we calculate the ucb_a^t,

  $$ucb_a^t = \widehat{e_a^t} + \sqrt{\frac{c \ln t}{u_a^t}}$$

  where $\widehat{e_a^t}$ is the mean reward of arm a until time t and the other variables have their standard meanings. It makes sense to substitute the empirical probability $\widehat{p_a^t}$ of 1 rewards, with mean reward $\widehat{e_a^t}$ because we are aiming at maximing the reward and minimising the regret.

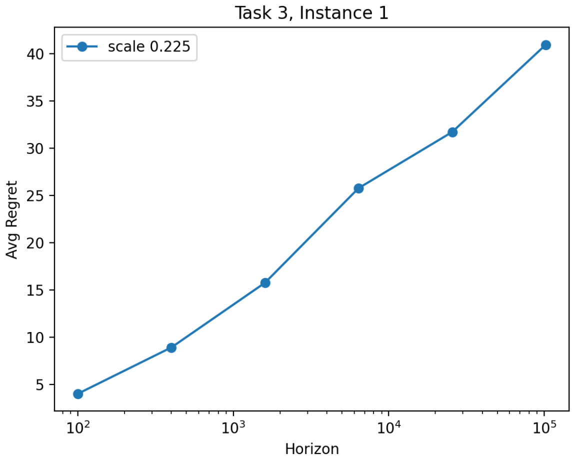
  Given that the UCB algorithm achieves a logarithmic regret w.r.t the horizon, we expect our data to generate a plot which shows nearly linear relation between the regret and the logarithmic horizon.

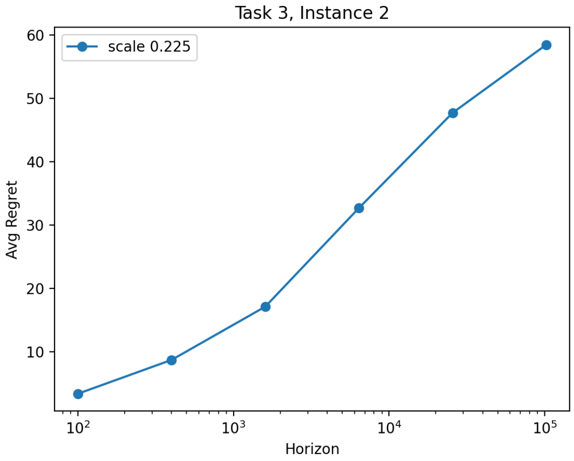
- **Algorithm:** The algorithm is as follows-

  - for us to begin with determining the ucb_a^t values for every arm, we pull each arm once
  - at time $t \geq N$, pull the arm with maximum value of ucb_a^t
  - `np.argmax()` is used for the above task, which resolves ties by returning the lower index

  I have adapted the UCB function created for the **Task-1** to work with such arms and bandits too. Hence, you won't find a separate function for this task.

- **Results:** The graph turns out nearly linear which is exactly what was expected. After several attempts, the optimal value for the scale, c was found out to be 0.225.





Task 4 - Maximising HIGH rewards

- **Overview:** The bandits in this task are of the same type as that of the **Task-3**. However, we have a major change in the problem as we look to maximise the **HIGH** rewards, where every rewards higher than a threshold value. th is considered **HIGH** and other rewards being **LOW**.

  This change basically makes our multi-reward arm nothing but a Bernoulli reward arm which we have implemented earlier. We have also seen that **Thompson Sampling** performs best when the rewards are binary.

- **Algorithm:** We will consider a **HIGH** as 1 and a **LOW** as 0.

  - for all the arms in the bandit, we sample a value from $Beta(\text{HIGH}_a^t + 1,\ \text{LOW}_a^t + 1)$, where HIGH_a^t and LOW_a^t is the total **HIGH** and **LOW** rewards respectively, for arm a until time t.
  - pull the arm with the maximum sampled value
  - `np.argmax()` is used for the above task which resolves ties by returning the lower index

The total reward i.e the total number of HIGHs are calculated as the sum of the HIGHs produced by each arm. For the regret calculation, we find out the arm with maximum probability for HIGH reward, call it p^*.

Let s be the support list and p be the probabilities list of arm a. Hence, the probability for HIGH reward of arm a,

$$p_a^{high} = \sum_{i;\ s[i]>th} p[i]$$

and thus,

$$p^* = \max_{a \in A}\{p_a^{high}\}$$

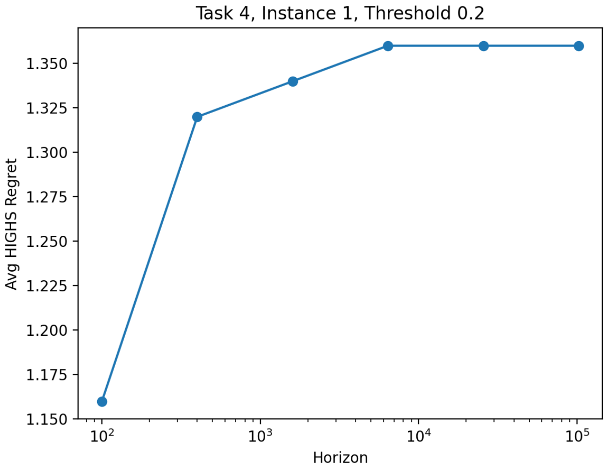
The regret is given by

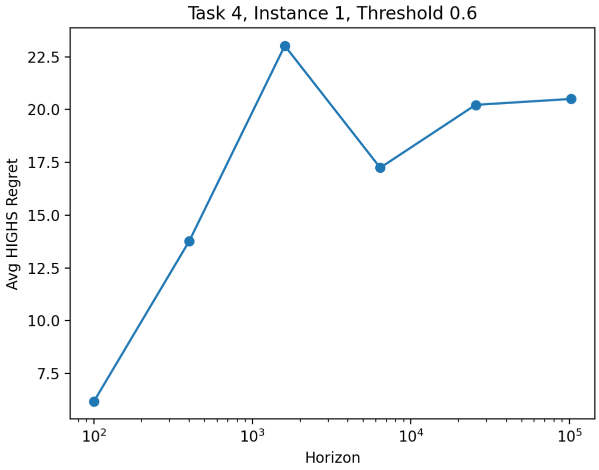
$$\text{regret-high} = p^* T - \text{HIGHs}$$

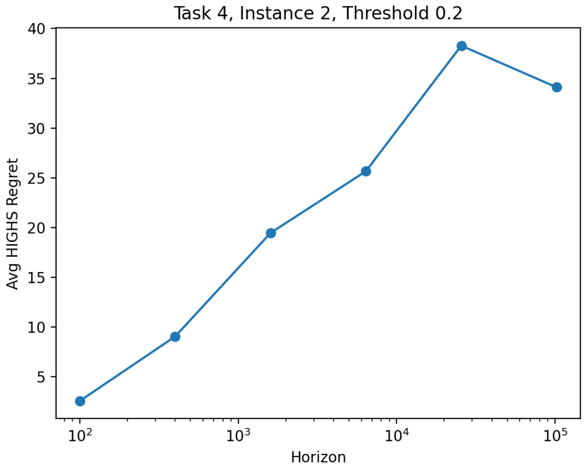
where T is the horizon.

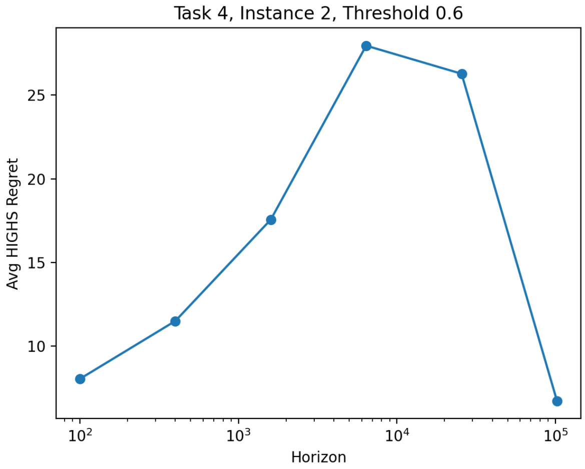
- **Results:** For threshold of 0.2, we see that the ration of HIGHs count to the horizon, T is very high. This is expected as the threshold is very low. For threshold of 0.6, in the first instance, we observe an increasing trend in the curve, and for the second instance, we find a striking dip in the curve with larger horizons.

  Overall, the ratio $\frac{\text{HIGHs}}{T}$ is very high for both the cases in each instance which also verifies that Thompson Sampling works brilliantly for binary rewards arms.


Task 4, Instance 1, Threshold 0.2


Task 4, Instance 1, Threshold 0.6


Task 4, Instance 2, Threshold 0.2


Task 4, Instance 2, Threshold 0.6