# CS 747 Assignment 1 Report

Shashank Roy - 180050097

Note:

a) Regarding all calculations, plots and data in **outputData.txt**, all values have been rounded to 4 places.
b) A **pull** refers to a function in code which samples reward from arm by simulating a pull.

## Task 1

For this task the multi-arm bandit is simulated by a list of **Bandit** class objects. Class **Bandit** is initialized by passing the **mean** of the instance.On simulating a pull on a bandit arm, the class variables are updated after generating reward from bernoulli distribution and the function returns the reward.

For each algorithm regret is calculated by subtracting generated reward from max-reward (max mean * horizon)

**epsilon-greedy-t1:**

For this algorithm I have used the **eG3** variation with $\varepsilon = 0.02$. In each time step, it first generates a decision variable with it's probability of being equal to 1 = $\varepsilon$.
If decision = 1 : then we select a random arm uniformly and **pull** it.
Else: we get the empirical mean of all the arms (rew generated uptil now / number of arm pulls). We find which arm has highest empirical mean and **pull** it.

**ucb-t1:**

In this algorithm we first initialize each arm with one **pull**. After that we do the sampling according to the standard UCB algorithm. We find the **ucb** for each arm and check which arm gives maximum and we pull that arm.

**ucb** for each arm is given by:

```
ucb = self.getempmean() + math.sqrt(2*math.log(step)/self.num_pulls)
```

Where **getempmean()** gets the empirical mean of the **Bandit** arm.

**kl-ucb-t1:**

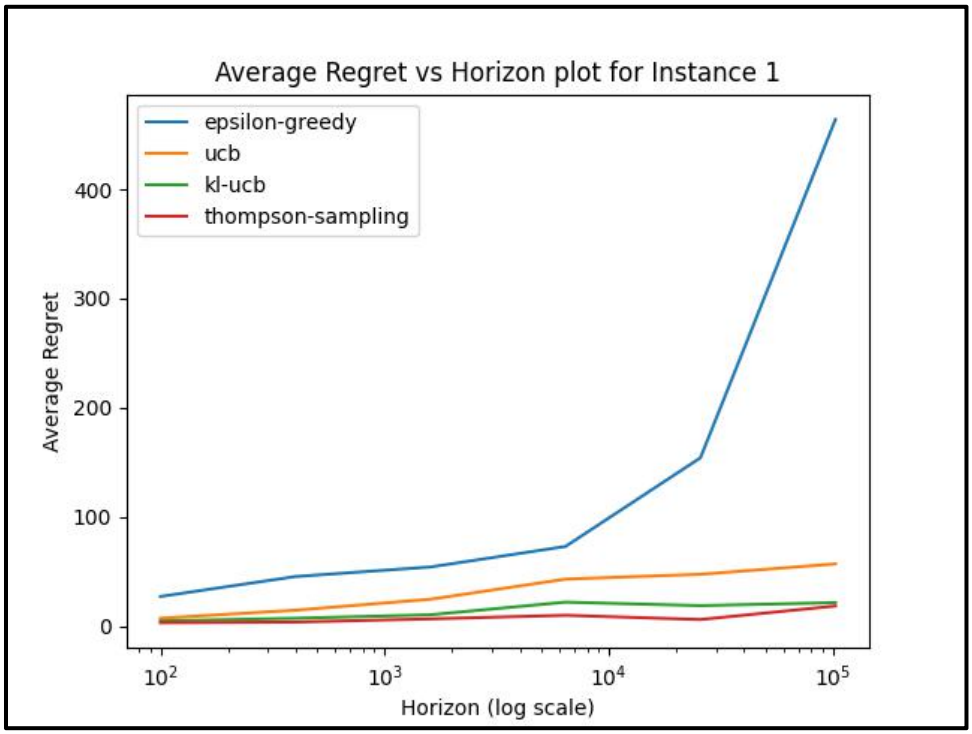
In this algorithm we first initialize each arm with one **pull.** After that we do the sampling according to the standard UCB algorithm. We find the **kl-ucb** for each arm and check which arm gives maximum and we pull that arm.
For calculating this we do binary search for q in range [empirical mean , 1] to maximize **q** while fulfilling the inequality constraint.

**thompson-sampling-t1:**

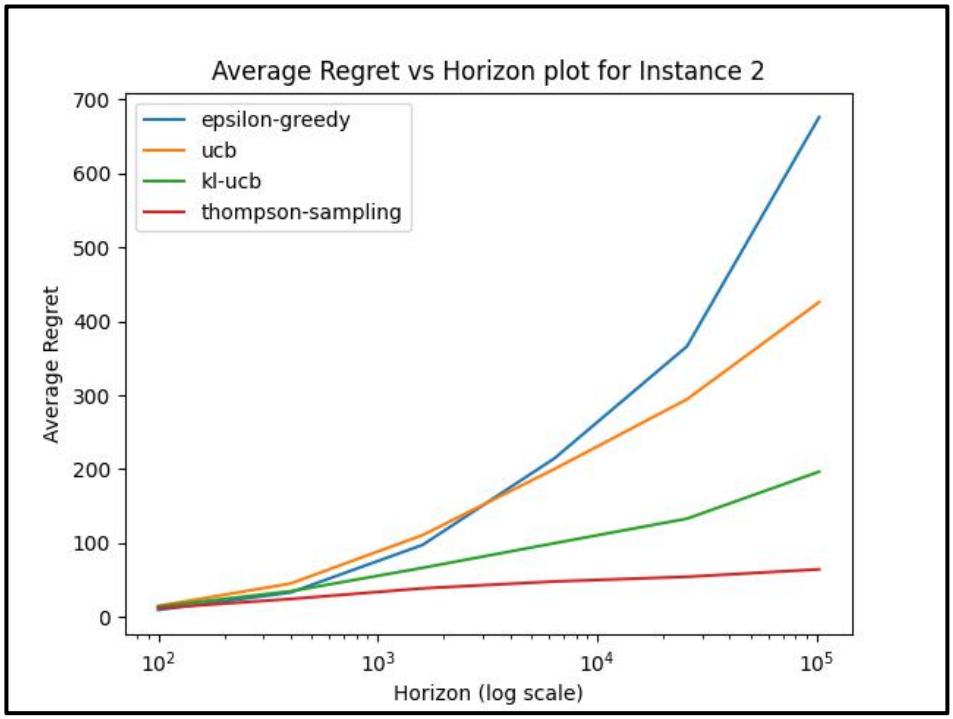
For this we simply **pull** the arm which gives the highest sampling value from beta distribution.

```
beta_sample_list = [np.random.beta(
           x.reward+1, x.num_pulls-x.reward+1) for x in multi_band]
```

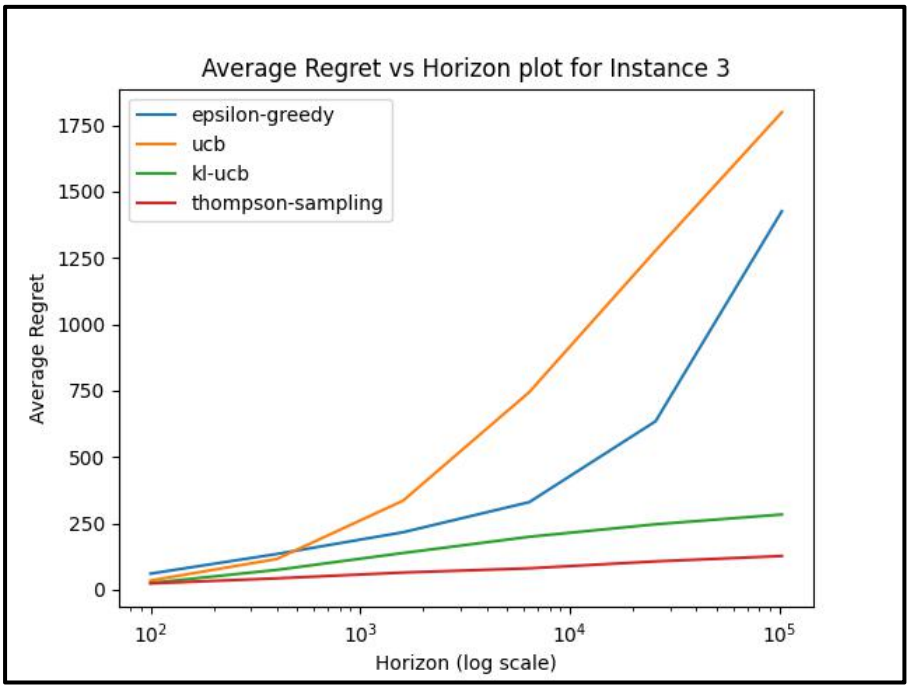
**Instance 1**



## Instance 2



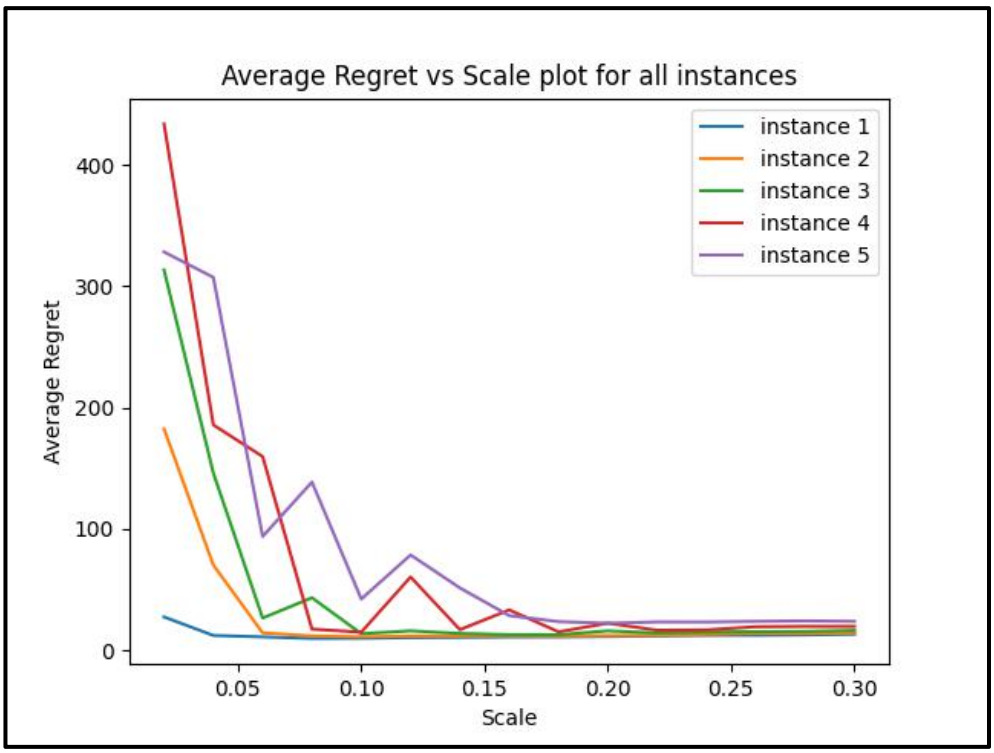

## Instance 3




**Observation:**
As expected the following trend is observed:
REG for thompson < kl-ucb < ucb < eG3

## Task 2

In this task it is observed regret has a decreasing trend by increasing scale **c**




| Instance | Optimal Scale |
|---|---|
| 1 | 0.08 |
| 2 | 0.1 |
| 3 | 0.18 |
| 4 | 0.1 |
| 5 | 0.2 |

The above reported values contain some randomness and hence Python gives us the 1st scale we find which minimizes the regret.

However the graph shows there is an overall decreasing trend of regret across all the instances. This is because for lower **c** the algorithm just uses the empirical mean to get the new UCB and decide which arm to pull.
On increasing **c** we enforce the algorithm to explore all the arms more intensely since an arm not pulled for some time will have a higher chance of getting pulled due to increased exploration bonus.
Hence it lowers the regret on account of better exploration and exploitation.

# Task 3

For this task I adapted the code from Task 1 & 2 for **Bandit** class to create a new **SupBandit** class which takes a support and probability vector for initialization. All the other functions like mean, calculating UCB remain same. But during a pull, reward is generated from the support vector with probability in the probability vector of bandit instance.

Let support vector be $s = [s_1, s_2, ...., s_n]$

Let probability vector be $p = [p_1, p_2, ...., p_n]$

Expected reward for one pull of bandit instance is calculated by:

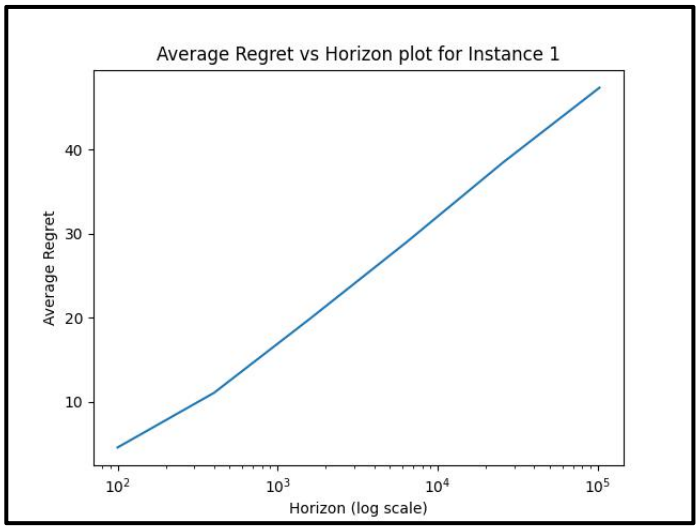
$$r = \sum_{i=1}^{n} p_i * s_i$$

Therefore the best expected reward for one pull $r^*$ is given by max expected reward of a bandit instance.
For each pull instead of adding only 0 or 1 in reward we add the reward generated from the support class.

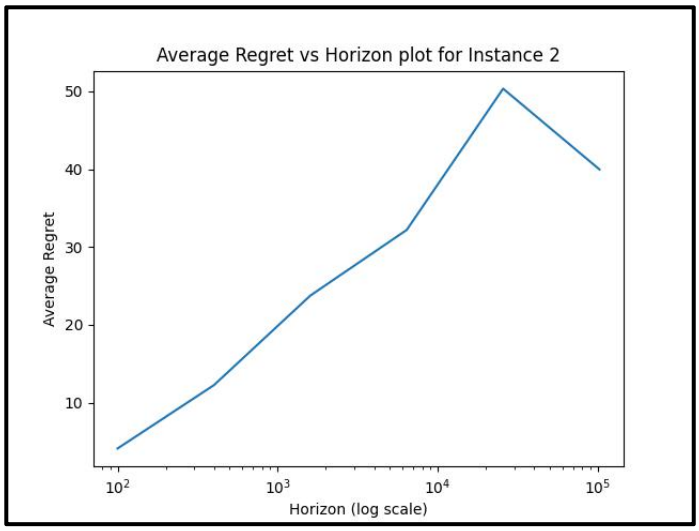
Let generated reward at step t = $r_t$

Hence REG = $\sum_{t=1}^{hz} r^* - r_t$

**Instance 1**



## Instance 2




As we can see the regret is quite low even for very large horizon (~ 40 - 50). Seeing the problem we can say that as an experimenter we would still be just pulling an arm and generating reward (which we want to maximize).

All the algorithms **pull** the arm, generate reward and process it. The same goes for this task.

The algorithm uses the Thompson Sampling algorithm to pull the arm. The algorithm requires number of successes and failures in order to generate sample from Beta distribution to check which arm to pull next. This is simply replaced by the cumulative reward of bandit instance produced up to now and for failure, (total pulls - cumulative reward).

I verified this algorithm by generating combined plots with other Task1 algorithms in addition to eG1 and eG2 algorithms.Also we know thompson sampling gives a very low bounded regret.

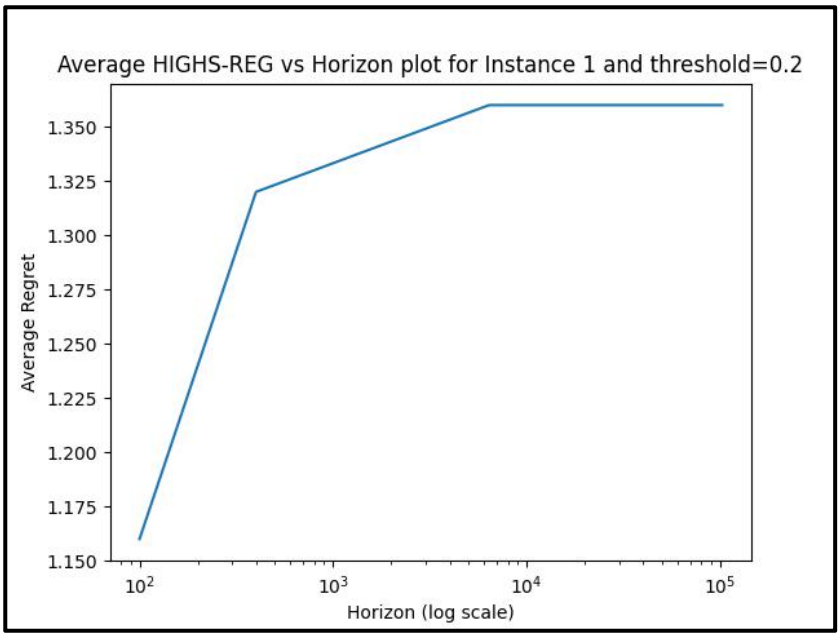
# Task 4

In this task we again use the **SupBandit** class defined above. But, during a pull instead of returning and storing reward I have compared the generated reward from the support with the passed **threshold** and store the result in **high** variable of class. If it exceeds then the pull returns 1 else 0 (accordingly adds 1 or 0 to the **high** variable).

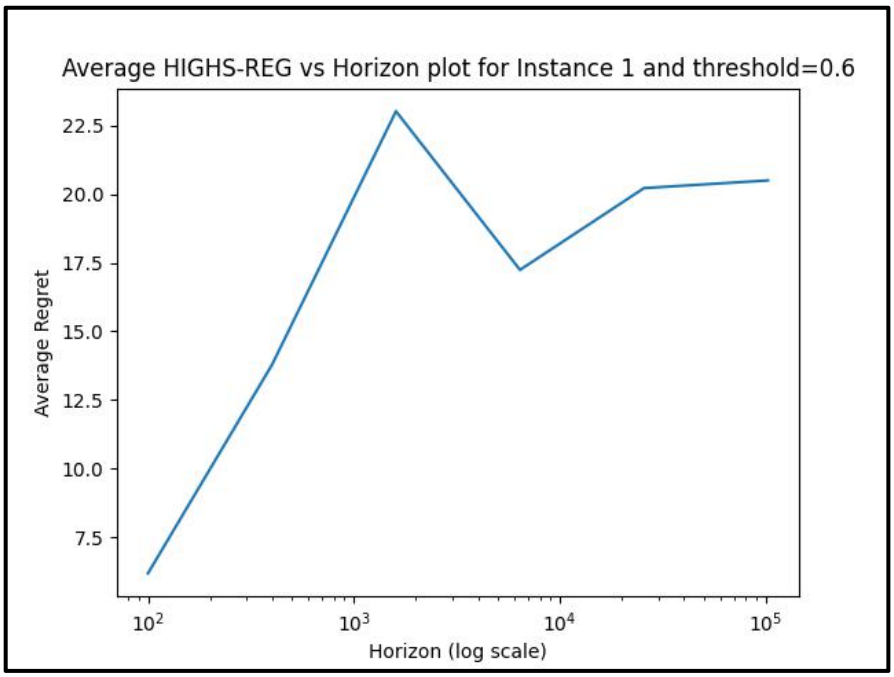
For plotting HIGHS-REG, MAX_HIGHS is calculated as follows:

- First extract the index of support values which exceed the given **threshold**.
- Then add the probabilities of corresponding index to calculate probability that a particular bandit instance produces a HIGH.
- Finally max of these probabilities give the probability to produce a HIGH when optimal arm is pulled.
- Finally multiply with horizon to get MAX-HIGHS

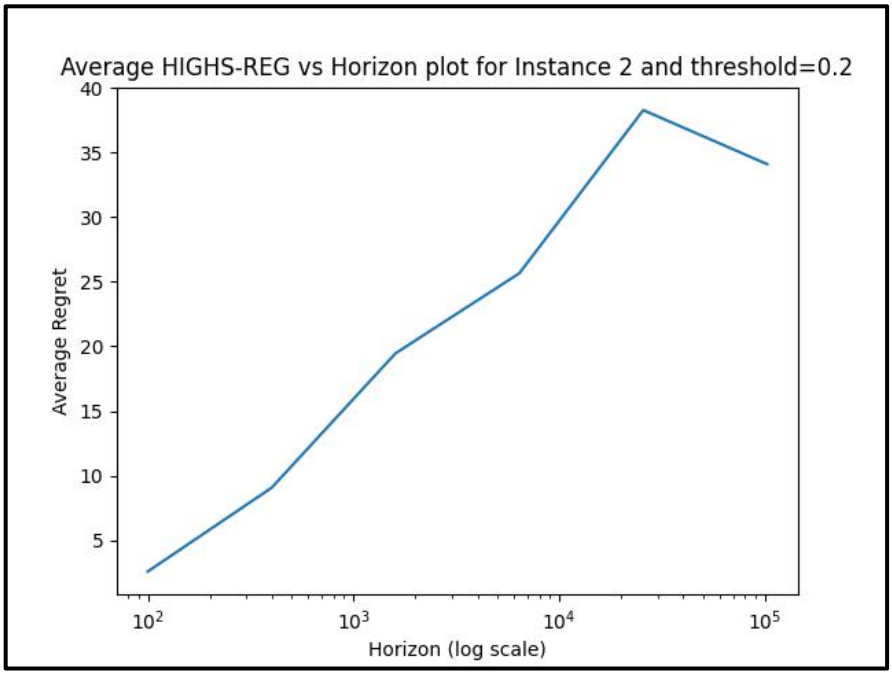
**Instance 1 and threshold = 0.2**



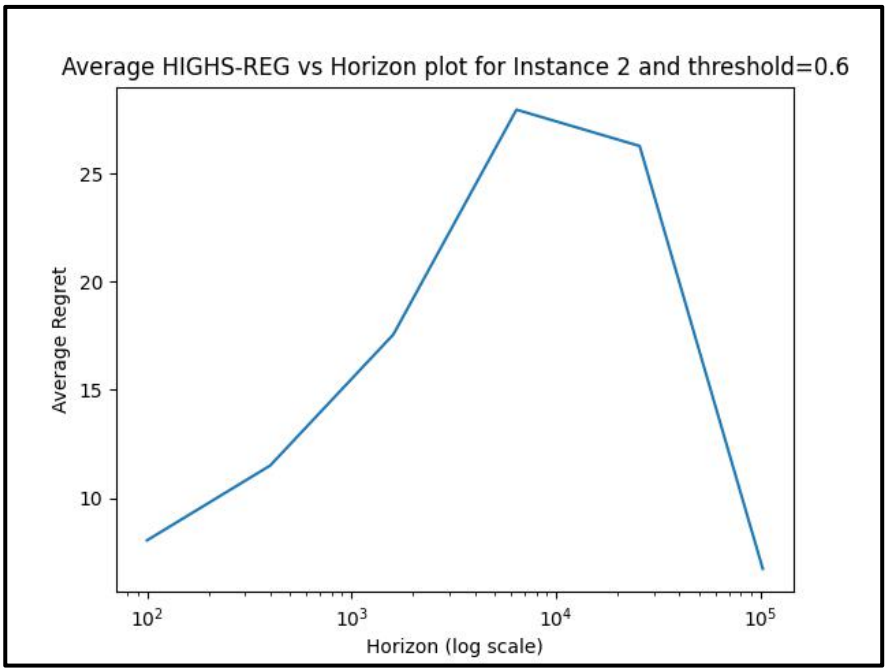
**Instance 1 and threshold = 0.6**




**Instance 2 and threshold = 0.2**



**Instance 2 and threshold = 0.6**




As we can see the regret in all these plots is very low (not even exceeding 40).
We may wonder why HIGHS-REG for **th = 0.6** is lower than **th = 0.2**. This is because **th** also affects the calculated MAX-HIGHS which is lowered for high **th** (since lesser values can be treated as HIGHS so expected probability of generating HIGH decreases).

Approaching this problem statement it is said to maximize the number of times we get a HIGH. This is analogous to saying we want to maximize number of success of this event.

Hence we once again use an algorithm inspired by Thompson sampling. For this we redefine success as the number of times we get a HIGH after a pull (REW > threshold) and failure as (total pull of arm - number of HIGH of arm).

I verified this algorithm provides minimum regret by sampling with eG3 and UCB.