CS 747 : Foundations of Intelligent Learning Agents
Assignment 1

Arka Sadhu - 140070011

August 14, 2017

1 Epsilon-Greedy

The epsilon-greedy algorithm works as follows :

- We choose some ε in the range $[0, 1]$.
- Then we choose the bandit with the highest empirical mean with probability $1 - \varepsilon$ and with probability ε sample an arm at random.
- ε is a constant given by the user. For all experiments, ε is chosen to be 0.1

2 Upper Confidence Bound (UCB)

The upper confidence bound (UCB) algorithm works as follows :

- We first pull each arm once in a round robin fashion.
- Then we compute the empirical mean of each arm. This is followed by an additional term which then gives the ucb for the corresponding arm.

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

- At each instance we choose the arm with the highest ucb value.

3 KL-UCB

This is the KL version of the UCB and works as follows :

- We again pull each arm once in a round robin fashion.
- Then for each arm we define a parameter q_a such that $q_a \in [\hat{p_a}, 1]$ and it is the least real number to satisfy the inequality 1

$$u_a^t KL(\hat{p_a^t}, q) \geq ln(t) + cln(ln(t)) \tag{1}$$

- We then choose the arm with the highest q_a
- Since KL is a monotonically increasing function, we employ binary search algorithm to search for q_a. We put the threshold for the above inequality to be $1e - 6$.

4 Thompson Sampling

Thompson Sampling algorihtm works as follows:

- We start by pulling each arm once in a round robin fashion till each arm is sampled once.
- Then we note each success and failure for a particular arm. Then we generate a beta distribution whose parameters are $\alpha = success + 1$ and $\beta = failures + 1$.
- We then sample a number x from the generated beta distribution for the corresponding arm and choose the arm with highest number sampled.

5 Graphs

5.1 How the Graph has been plotted?

For each instance, for each algorithm, the script is run 100 times with horizon = 10^5. The server side of the code is slightly modified to return the regret at each horizon. Therefore we have the regret at each time instance t and we average over the regrets at each time instance t to get the final plot. The x-axis is plotted in log scale to show the linearity of three of the algorithms. Also the actual Assignment task asks us to plot for horizon = 10, 100, 1000, 10000, 100000 and I felt it would be appropriate therefore to plot in the log scale.

5.2 Actual Plots

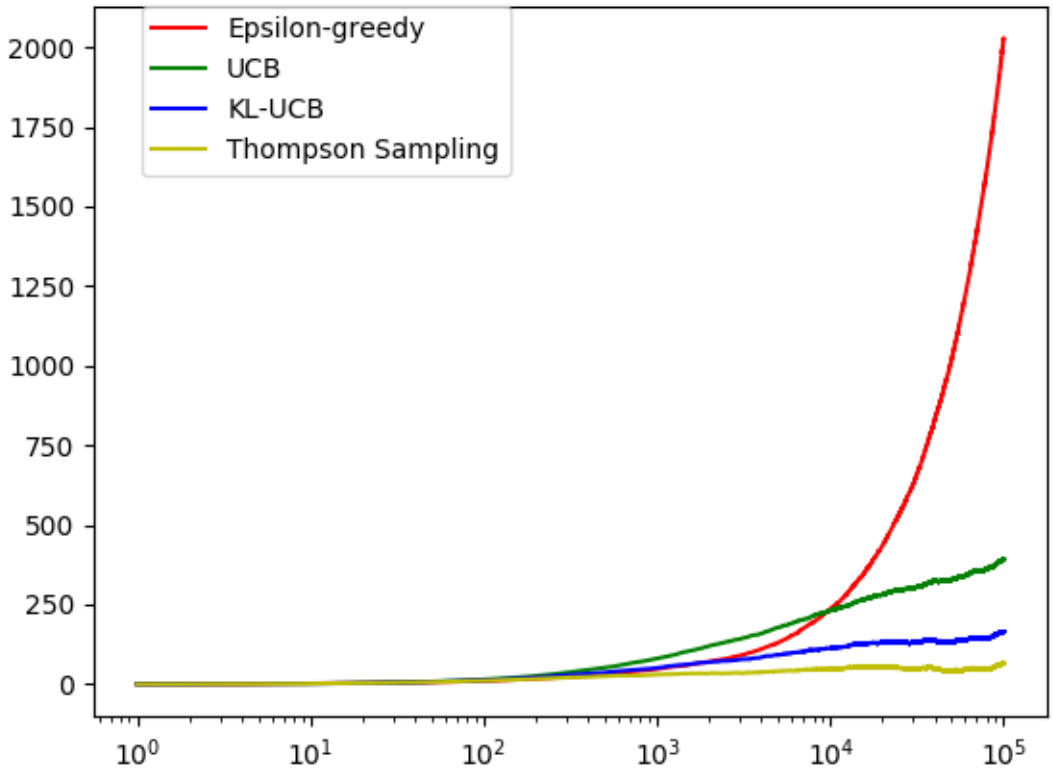

Figure 1: Instance-5 All Algorithms

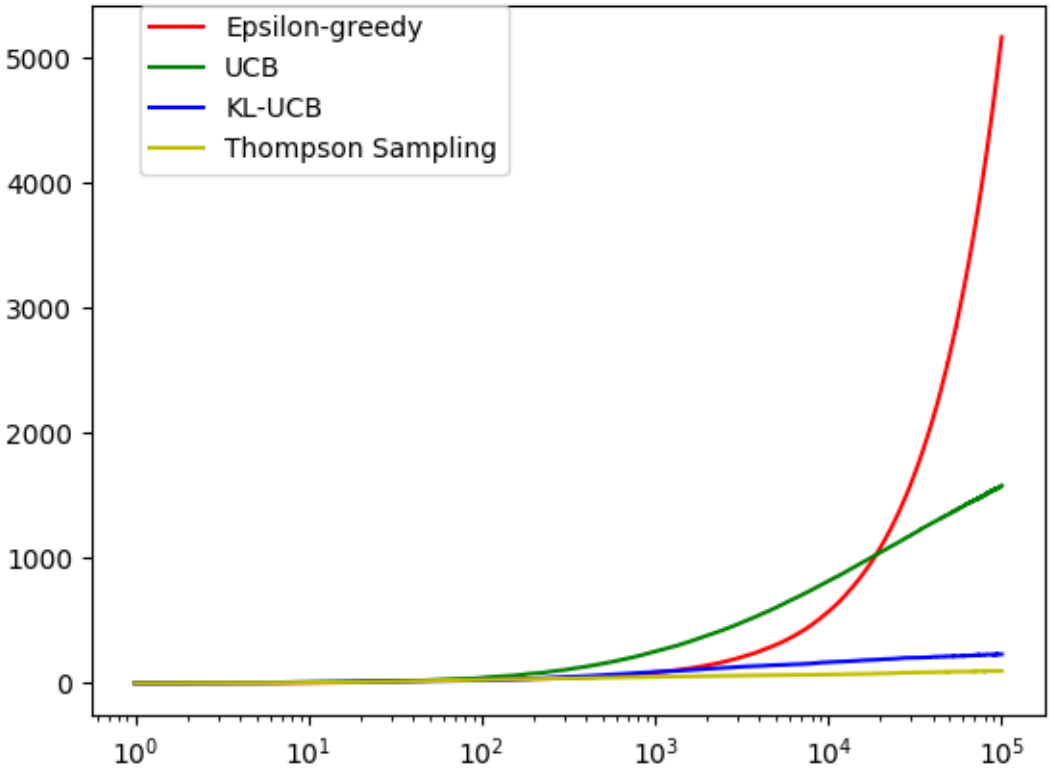

Figure 2: Instance-25 All algorithms

6 Observations

6.1 Timing difference

Interestingly there is quite some difference in the time taken by the algorithms for execution, and it is more apparent when the horizon is 100000. In ascending order of time taken for completion of one run with horizon = 10^5

$$\varepsilon - greedy < ThompsonSampling < UCB < KL - UCB$$

. Whereas in ascending order of regret it is (as can be seen from plots 1) and 2)

$$ThompsonSampling < KL - UCB < UCB < \varepsilon - greedy$$

So it apparently seems Thompson Sampling does significantly better than the remaining algorithms.

6.2 Log linear

The plots in 1 and 2 clearly show that the Thompson Sampling, UCB and KL-UCB display linearly varying regret with respect to the log of the time instance. This is consistent with the theory taught in class. It is difficult to ascertain the relation of $\varepsilon - greedy$ but it does seem to be sub-polynomial in time (noting the linear plot in 3)

6.3 Negative Regret in some cases

As noted earlier, the server side of the code is slightly modified to return the regret at each time instance. It was observed that for quite a few random seeds and at quite a few time instances the

regret was shown to be negative. This is interesting because at first sight it seems that we are actually doing better than the best case possible. But it turns out that when averaged over all the runs the regret is positive [at any specific time instance], implying that the negative regret was mainly because of the stochastic nature of the server side.

6.4 Choosing suboptimal arms at lower horizon

It is noted that for each of the algorithms for instance-5 and horizon = 400, the optimal arm is not always the one which is pulled maximum times. For example in the case of random seed 0, and horizon 400, the UCB algorithm pulled arm 3 (arm numbers indexed starting from 0) whose actual probability of success was 0.4 while the optimal arm was in fact arm 4 whose probability of success was 0.5. It turns out that increasing the horizon to 10000 and above solves this problem and the algorithm eventually starts choosing the optimal arm.

6.5 Jittery graph for instance-5

At first sight, the jittery graph seems a bit odd for instance-5, especially because it implies that regret could actually become less with more horizon, which is not true intuitively. In plotting the graph, 100 runs of each algorithm have been run, and regret at each instance is marked. So the only plausible explanation is that the decreased regret is only because of the stochastic nature of the experiment.

6.6 Choosing the maximum

This is less of an observation, more of an implementation detail. In dealing with ucb of individual arms, we have to break discrepancy when two arms have the same maximum value. In the code, the first occurence is always chosen. It is noted that making this deterministic doesn't really affect the regret on the average.

6.7 Epsilon greedy doing better than UCB when horizon is small

It is noted that for small horizons, Epsilon greedy approach (with $\varepsilon = 0.1$) actually outdoes UCB algorithm as can be seen in 3. The reason for this is presumably the reason that UCB algorithms pull the sub-optimal arms at smaller horizons and in general requires more time for exploration, whereas exploration in $\varepsilon - greedy$ is done stochastically and happens much faster. The problem that persists with $\varepsilon - greedy$ approach is that even after sufficient confidence about the optimal arm, it still tries to explore though it is exploration is not actually required. UCB circumvents this by creating the upper confidence bound, and restricts random explorations.

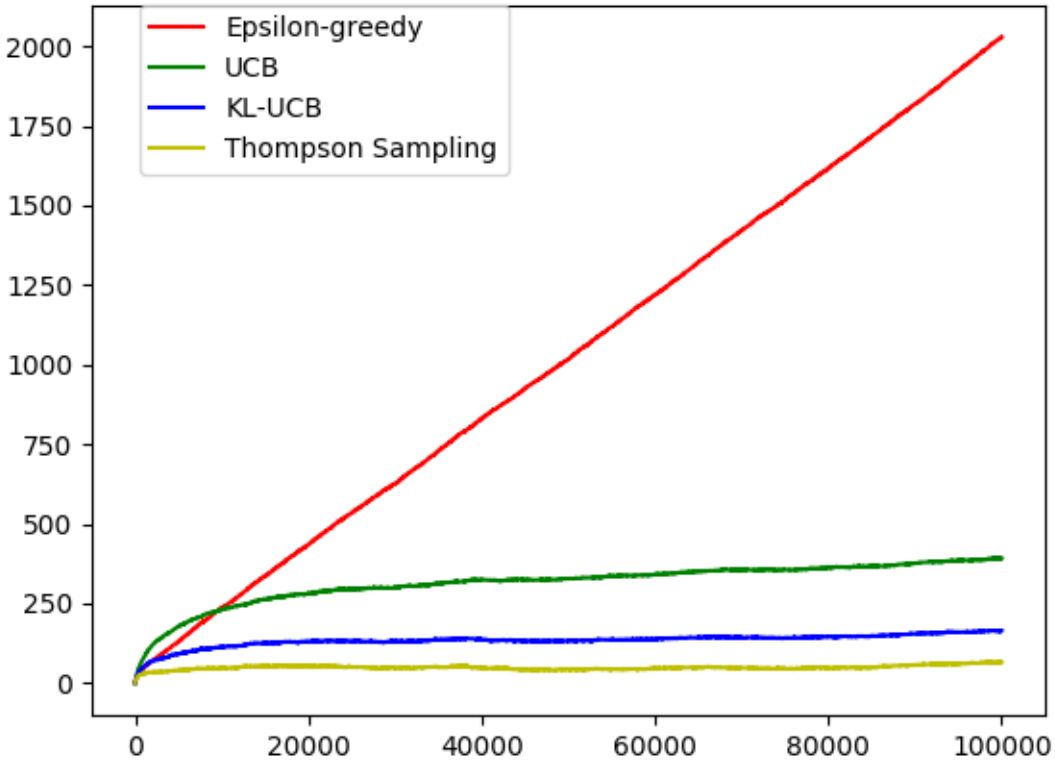

Figure 3: Linear Plot of Instance-5

7 Libraries used

The client side is coded in python. The only modules used are **Numpy** for random number generation, **socket** module for socket communication, **re** module to scrape the regret values, **Matplotlib** for graph plotting and **pdb** for debugging. The code has been run and tested on sl2 machine prior to submission.