CS747: Assignment 1 Report

Manan Sharma (170040067)

September 25, 2020

Task 1

This task consists of implementing 4 algorithms (epsilon-greedy, UCB, KL-UCB and Thompson Sampling). The code is modularized, and each algorithm resides in a different file of same name as algorithm, invoked only by *bandit.py*. Some algorithm specific details are:

Epsilon-Greedy: The eG3 variant discussed in the lectures is implemented, which explores at any timestep with probability ε (provided by the runner), and exploits by pulling the arm of highest empirical mean with probability 1- ε. As proved in lectures, this algorithm does not achieve a sub-linear regret.

In beginning, the algorithm implemented **does not** follow round-robin and starts exploring/exploiting as per above scheme right away. Ties happen, especially a lot in the initial timesteps, while the exploitation phase, where sometimes, there are multiple arms having the highest value. In that case, we simply pick first such index in the maintained array of empirical means.

UCB: An auxiliary function for calculating the UCB function (as defined in the lecture slides) is implemented, which, at each timestep, returns the ID of the arm with highest value of the UCB function, and that arm gets pulled. Since the UCB function involves division by number of pulls of an arm, it was therefore **necessary** to make first pulls in a round robin fashion, to avoid division by zero. Albeit, only a single pass of round robin happens, after which the UCB algorithm follows. For breaking ties, the arm with maximum index in the array is pulled.

KL-UCB: The KL-UCB function (as defined in the lectures) requires us to calculate the maximum value of q, that satisfies the given inequality which in turn requires computation of the KL function. The value of **c** in the RHS of the inequality is set to 3. Now, since the RHS and an arm's empirical mean are a constant for a given timestep and KL(p,q) is an increasing function of q for a fixed p, a binary search is carried for finding out a suitable value of q with a suitable accuracy (in implementation, the accuracy is set to 1e-3). So the resulting q is guaranteed to be within at most 1e-3 in left side of the optimal value of q (point where LHS=RHS). The conditions prevent q from becoming greater than the optimal q.

Also, since the LHS in KL-UCB function's inequality involves number of pulls, a single pass of round robin is carried out to avoid the case of LHS becoming zero. For breaking ties, the arm with maximum index in the array is pulled. Notably, KL-UCB took highest running time among all algorithms, keeping other arguments fixed.

Thompson Sampling: Due to the settings of sampling from Beta distribution, there was no need of round-robin since when the arm isn't pulled at all, the Beta belief is simply a uniform distribution. So, no special mechanism is used in first pulls. For breaking ties, the arm with smallest index is pulled.

The following plots demonstrate the behaviour of algorithms in each of the instances. These are made directly using the outputDataT1.txt file. The average (on the y-axis) is taken with respect to 50 seeds.

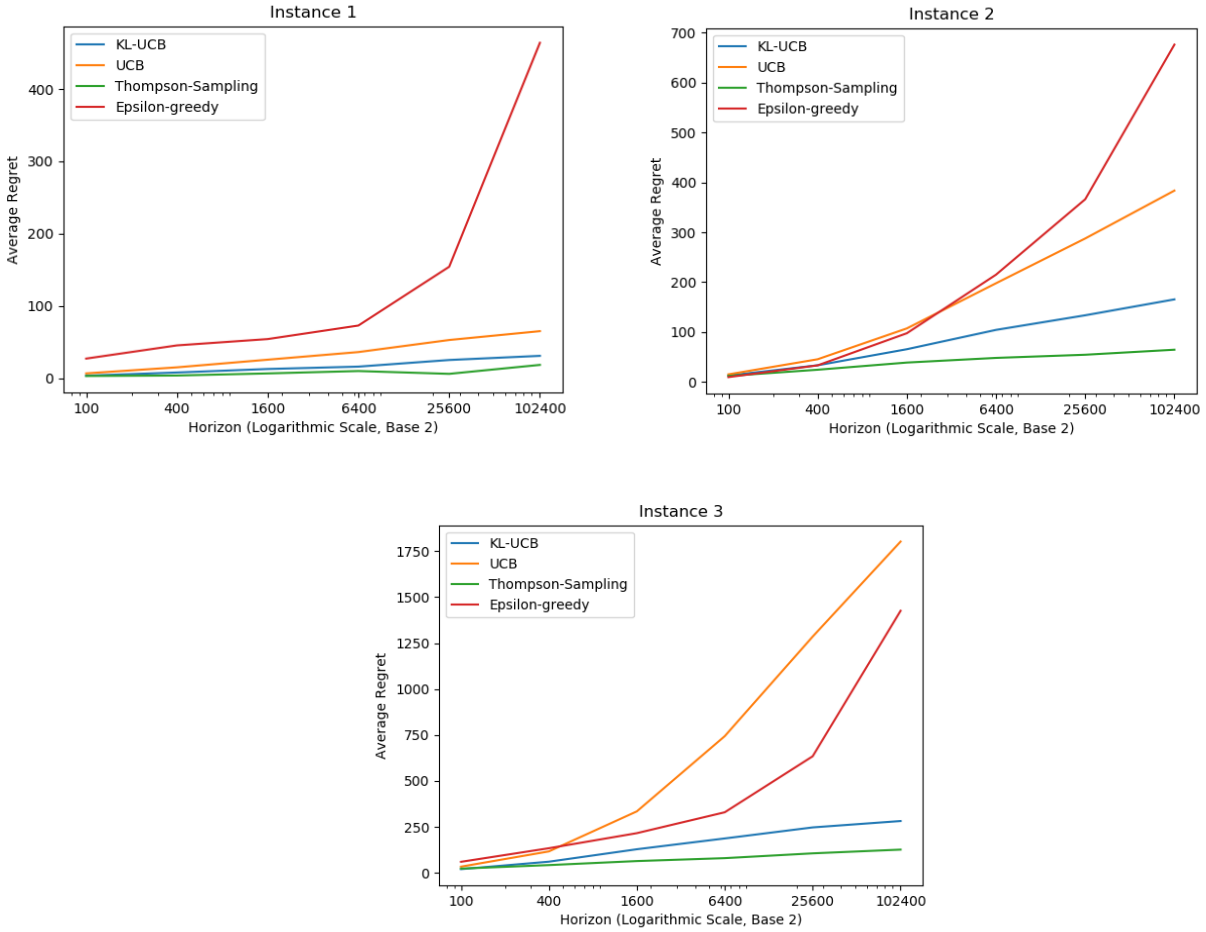


Figure 1: Plots for each of the instances, for task 1

Observe that in plot of instance 3, UCB shows highest regret. But as per the trend in plot, epsilon-greedy has a higher slope and eventually seems to surpass UCB

Task 2

This involved performing Thompson Sampling with additional information given, which was the permutation of the true mean assumed to be given in a descending order. Several ideas were tried and some are listed below:

1) One idea was to extract the maximum of the true mean, and set the initial parameters of the beta distribution in each arm such that, the initial belief of means of all arm is exactly the true mean of optimal arm (i.e. a = num_rewards[arm] + factor*p_max +1 and b = num_zeros[arm] + factor*(1-p_max) +1). The factor is a user defined constant that measures the strength of the belief, low values are preferable. But a lot of information goes wasted here. Also, the beliefs do not hold true for most of the arms. On an average, this performed just like Thompson Sampling, or sometimes giving a little more average regret than the former.
2) Other idea was to maintain a changing prior, in which, instead of using the obtained number of rewards and number of zeros/fails, simply use the value from the given array of true means, that is closest to the current empirical mean (i.e. a = factor*p_i + 1 and b = factor – factor*p_i + 1 where p_i is the closest element of the given true means array to the current empirical mean of arm i). Here, factor is a quantity that linearly increases with number of pulls, since as the arm gets pulled more and more, the confidence interval

decreases and belief is strengthened. This however has the issue that it gets stuck in the suboptimal arm and doesn't explores much.

Considering all this, following is implemented:

Make the parameters a and b dynamic, which simultaneously incorporate the information given as the permutation of true means. That is, at any time step, for an arm i, sample from Beta(a,b) with a and b given as:

a = num_rewards[i] + factor*f(guess_mean, empirical_mean[i])*guess_mean +1

b = num_fails[i] + f(guess_mean, empirical_mean[i]) - f(guess_mean,empirical_mean[i])*guess_mean + 1

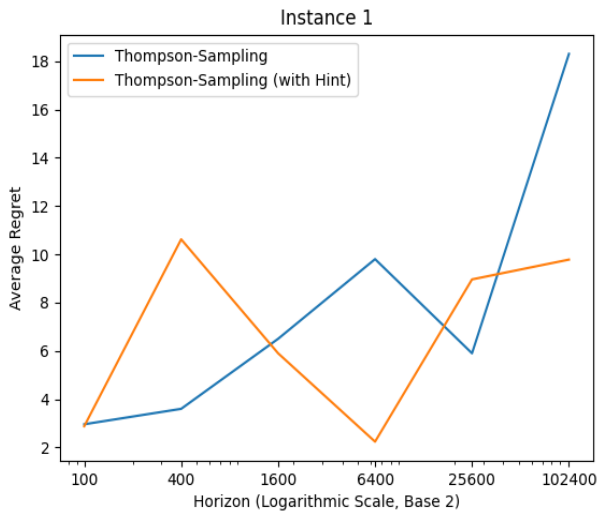
where num_rewards[i] and num_fails[i] is number of rewards obtained and number of fails respectively, guess_mean is a value of the true mean we guessed from the given list of true means from based on current empirical mean of arm i, empirical_mean[i]. The guess_mean is calculated as follows: Sort the current array of empirical means of all arms in descending order, find the position of arm i's empirical mean, and return the value present in the corresponding index of the given true mean array (which in our case, is sorted in descending order too).

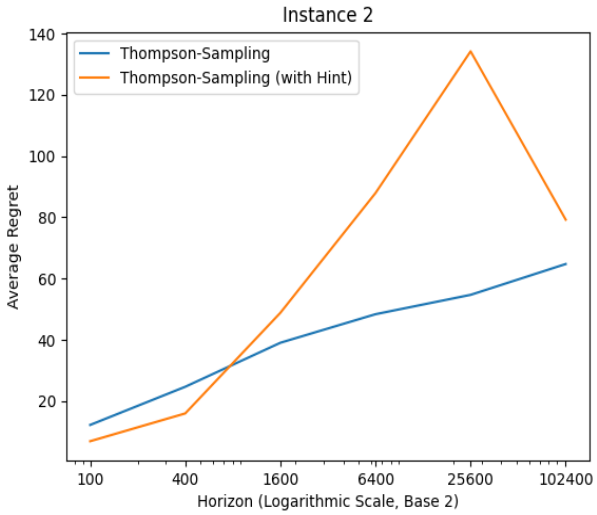
The function f is defined such that, its value is high when empirical_mean[i] is near to guess_mean and lower when they're farther. This essentially encodes the strength of belief we have. Many candidates of f() were tried (like one based on softmax of array with empirical_mean[i] subtracted throughout), and the most suitable one was:

$$f(empirical\_mean[i], guess\_mean) = horizon * e^{-|empirical\_mean[i] - guess\_mean|}$$

Multiplication by horizon is done only by experimental basis. Lastly, the term 'factor' present in a is set such that it decreases as more pulls are made. This was done because, as the pulls increase the confidence we have in empirical mean increases, so num_rewards[i] is alone sufficient to capture the mean. So, factor is simply set as 1/(num_pulls[i] + 1).

This does seem to work better than normal Thompson Sampling since during the analysis, this pulls the optimal arm a greater number of times than the normal Thompson Sampling in majority of the cases. Though in very rare cases, it gets stuck in suboptimal arm, but that is only observed for smaller horizons. Following are the plots obtained.





Continued.

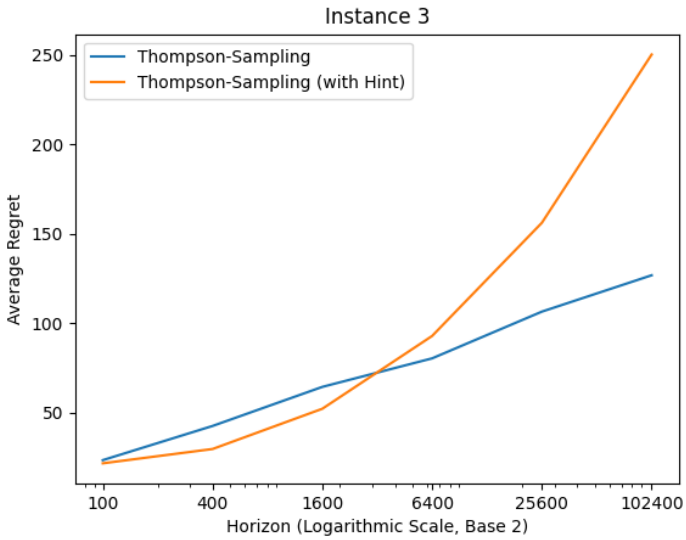


Figure 2: Plots for each of the instances, for task 2

Only for instance 3, its performance seems to deteriorates with horizon. One possible explanation behind this could be the greater number of negative regrets obtained for the normal Thompson sampling as compared to this one, in the experiment setting. For one seed in instance 3, the algorithm was indeed found stuck on the suboptimal arm. Maybe we can expect the curve to drop at some later horizon, like it does in plot of instance 2.

Task 3

The experiments were done on values of epsilon from 0.001 to 0.01 (both inclusive) with step size of 0.001. The findings are:

Instance 1: $\varepsilon_1 = 0.005$ (Regret = 312.74)

$\varepsilon_2 = 0.006$ (Regret = 298.2)

$\varepsilon_3 = 0.008$ (Regret = 322.8)

Instance 2: $\varepsilon_1 = 0.003$ (Regret = 2072.34)

$\varepsilon_2 = 0.004$ (Regret = 1628.2)

$\varepsilon_3 = 0.005$ (Regret = 1873.42)

Instance 3: $\varepsilon_1 = 0.004$ (Regret = 2581.8)

$\varepsilon_2 = 0.005$ (Regret = 1857.72)

$\varepsilon_3 = 0.006$ (Regret = 2005.96)

Each regret value was obtained after averaging for 50 seeds for each epsilon.

Task 4:

The plots are attached in this report in the relevant sections. Please find the required data in *outputDataT1.txt* and *outputDataT2.txt* files in the submission.

Additional Details:

1) The bandit instance is being passed into every algorithm, but no algorithm uses it. This was done because of an arm pulling function that resides in each algorithm's module, which requires the bandit instance. Other than that bandit instance isn't used (though its length is used in some places for getting the number of arms)
2) **For the TA:** The provided *verifyOutput.py* file is seeded with value 2. For this seed, if run on *outputDataT1.txt,* it would pick a line corresponding to KL-UCB for i-3 for horizon 25600. This might take up to a minute or a few minutes.
3) Only a minor error checking for input is done for the code, i.e. the code might behave in a weird way for any unexpected/abnormal input.
4) The ordering in the data files seems to be random since they were generated in a parallelized form in order to save time.