CS 747 Programming Assignment 1

Parth Shettiwar: 170070021

September 2020

Contents

1 Overall Implementation

There are total 9 functions in bandit.py, namely:

```
1) greedy
2) UCB
3) KLUCB
4) Thomson
5) Thomson_Hint
6) KL
7) klucb_opt
8) thomson
9) reward
```

The first 5 functions each corresponding to main call for respective implementation of each algorithm. Apart from that, there are 4 other functions, "reward" calculates the reward given an arm index, "thomson" samples values from beta distribution corresponding to each arm and returns the max arm index giving highest value, klucb-opt does the binary search required for kl-ucb algorithm and "KL" function returns the KL divergence between the empirical mean and q value. The whole code is written bandit.py and imports only numpy and sys. Also the bandit instance is made global and acceesed only by reward function for generating rewards. The rest of functions and algorithms access just the length of bandit instance or basically the number of arms.
The command line arguments are taken using sys.argv. Hence its important, the paramters are provided in correct order.

2 Epsilon-Greedy

- It is the implementation of Epsilon-greedy 3 startegy described in lectures
- Takes random seed, epsilon and horizon as paramters
- Round Robin once for all arms at start (to make non-zero pulls for each arm).
- For exploitation phase, if 2 or more arms have same highest empirical mean, pull the arm with lower index (tie breaker).
- Calls `reward` function

3 UCB

- It is the implementation of UCB algorithm described in lectures
- Takes random seed, epsilon(redundant) and horizon as parameters
- Round Robin once for all arms at start (to make non-zero pulls for each arm).
- If 2 or more arms have same highest UCB value, pull the arm with lower index (tie breaker).
- Calls `reward` function

4 KL-UCB

- It is the implementation of KL-UCB algorithm described in lectures
- Takes random seed, epsilon(redundant) and horizon as parameters
- c is set to 3, the constant while calculating q value. The tolerance for binary search is kept at $1e^-3$
- Round Robin once for all arms at start.
- If 2 or more arms have same highest "ucb-kl" value, pull the arm with lower index (tie breaker).
- Calls `reward` function and `klucb_opt`(for binary search for q value) which in turn calls `KL` function.

5 Thompson Sampling

- It is the implementation of Thompson algorithm described in lectures
- Takes random seed, epsilon(redundant) and horizon as parameters
- Used np.random.beta for modelling the distribution for beliefs for each arm.
- If 2 or more arms have same highest sampled value from their beta distribution, pull the arm with lower index (tie breaker).
- Calls `reward` and `thomson` functions.

6 Thompson Sampling with Hint

Since we have full knowledge of true means, the belief is computed only over the true means. The algorithm is as follows:

1. Take input *permutedmeans*, random seed, and horizon. For simplicity, have taken the sorted means increasing order as permuted means.
2. Initialise the belief for each arm over all true means i.e. (1/total number of arms) belief over each true mean for each arm. Initialise the successes and failures for each arm as 0. Success is the number of time arm gives reward 1 and failures is number of times arm gives reward 0.
3. Get the belief value for each arm corresponding to highest true mean and pull the arm corresponding to max of these belief values If 2 or more arms have same highest belief value for highest true mean, pull the arm with lower index (tie breaker).
4. Note the reward, update the success and failures for that arm. Update the belief vector corresponding to that arm. For a pulled arm A and a particular true mean x, the update for that arm will be:
   If reward is 1:
   $$Belief_A[x] = \frac{Belief_A[x] * x}{\sum_{y \in permutedmeans} Belief_A[y] * y}$$
   If reward is 0:
   $$Belief_A[x] = \frac{Belief_A[x] * (1 - x)}{\sum_{y \in permutedmeans} Belief_A[y] * (1 - y)}$$
5. Go to step 3 for horizon times.

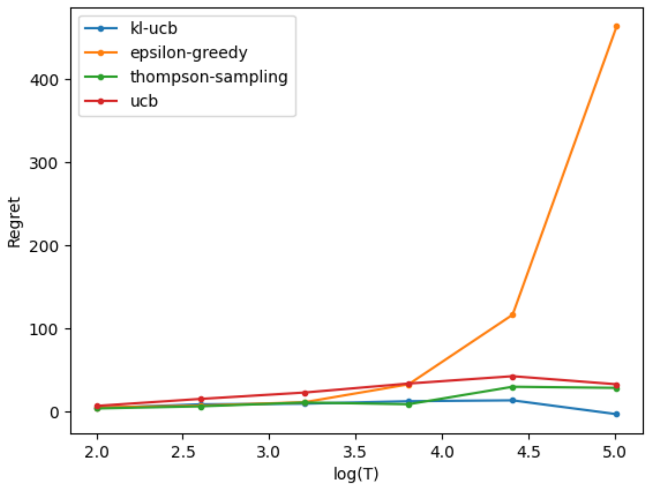
The underlying idea is the same as basic principles of Thompson sampling. Just instead of beta which is continuous distribution, we are calculating distribution over discreet values. However one slight change is unlike Thompson algorithm, instead of sampling from each arm, we are simply taking the belief value corresponding to highest true mean. This is because we want to focus our attention on arm which actually corresponds to highest true mean and we would minimize regret only when we pull that arm only. Normally we cant do this, since we don't know the total number of possible means. Hence we need to sample for each arm distribution in order to explore. Here we explicitly know the mean corresponding to optimal arm would be highest in the True means provided. This algo is definitely going to give sub-linear regret. Further the belief corresponding to optimal arm would be a one hot vector with 1 at the Highest True mean
$Proof$:
Firstly its based on the Thompson belief distribution principles, hence we know its going to give sub-linear regret definitely. Further now assume, if a sub-optimal arm gets the highest belief for highest true mean, then it would be picked continuously. However as Horizon $\rightarrow \infty$, the belief of that arm for its true mean would increase and subsequently the belief of that arm for highest true mean would decrease, dropping after some level, it would be now pick another arm with highest belief for highest true mean(effectively exploring again) and would do this continuously for other sub-optimal arms(in worst case) until it starts picking optimal arm continuously.

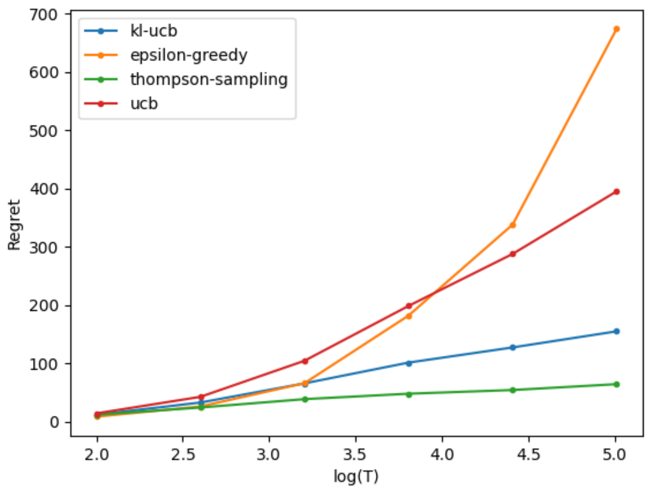
7 Results

7.1 Analysis of T1

7.1.1 Graphs

The base of 10 is used for log(T) as x-axis. The following are the graphs of Regret vs Horizon for the 3 instances(as average of 50 seeds) for the 4 algorithms:





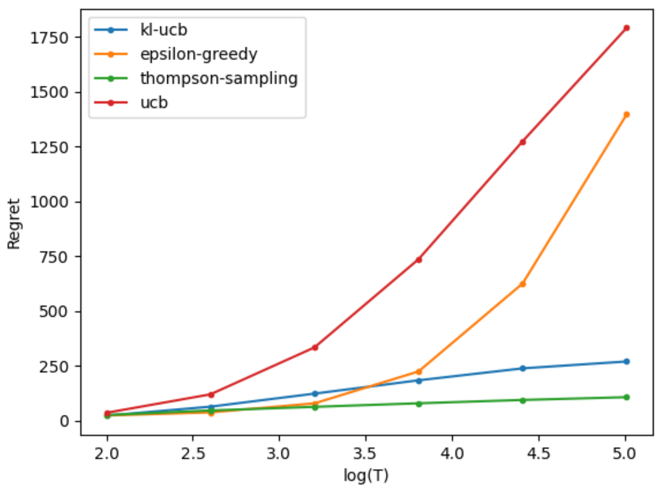


Figure 1: Top: Regret values for Instance i1.txt, Middle: Regret values for Instance i2.txt, Bottom: Regret values for Instance i3.txt. Note: The points are connected by straight lines(linear interpolation)

7.1.2 Observations

- At small number of bandits i.e. i1 in our case, the ucb, kl-ucb and Thompson sampling give very close results and for average of just 50 seeds, they dont get the curves which they should ideally. The epsilon-greedy gets exponential curve(for logT scale) which means linear regret which is theoretically proved. KL-ucb gets the least regret in end for long horizon for i1. However we know Thompson sampling is good in practise and should have got lesser regret. Since its just average of 50 seeds and less bandits, hence this can happen.
- At more than 2 bandits, i.e. i2 and i3, we see the ideal curves which we expect to come. The epsilon-greedy is exponential. KL-UCB, UCB and Thompson sampling get linear curves implying logarithmic regret. For i2, UCB>KL-UCB>Thompson Sampling. This is because in theory also,UCB does get logarithmic regret however it differs in constant for the lower bound and is more than KL-UCB and Thompson sampling.
- For i3, we observe one peculiar thing. Regret of UCB always greater than epsilon-greedy. However a closer inspection reveals, UCB curve is linear(barring for some short horizons) and epsilon-greedy is exponential. Hence we know, epsilon-greedy would surpass UCB at some later horizon value. The nature of curve matters and that is correct for all algorithms except in i1 where there are only 2 bandits(where higher chance of getting very low regret and not necessarily logarithmic, also average of just 50 seeds dont always reveal the correct nature of curve).

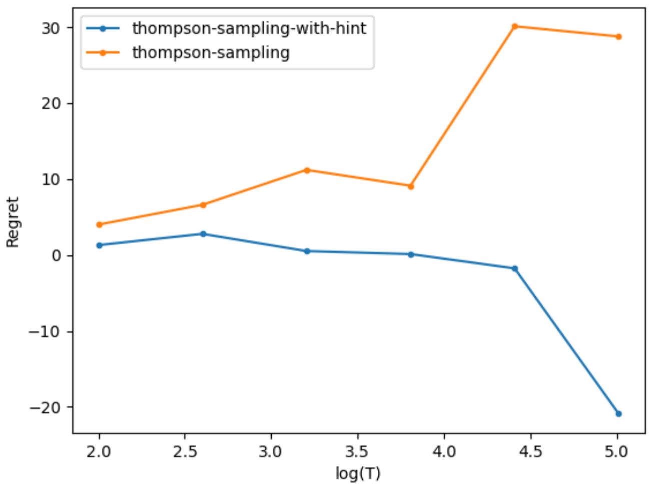
To summarise:

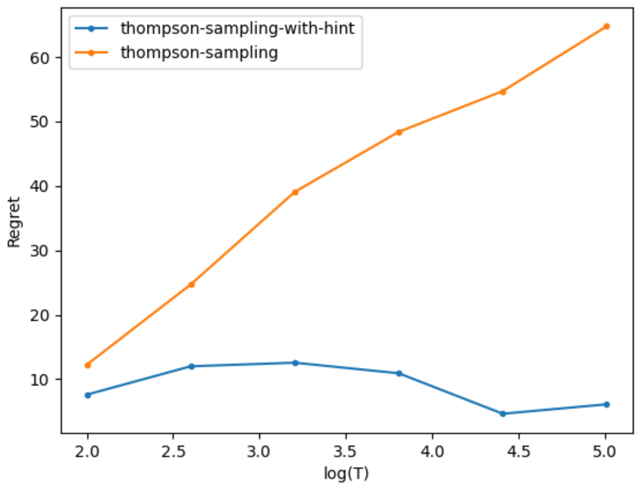
- Epsilon-greedy is always exponential
- For higher bandits,regret order is UCB>KL-UCB>Thompson Sampling and all achieve linear curve(Logarithmic regret on T scale) conforming to ideal cases.
- For small number of bandits, we get very less regret values and sometimes decreasing with horizon for above 3 algos.

7.2 Analysis of T2

7.2.1 Graphs

The base of 10 is used for log(T) as x-axis. The following are the graphs of Regret vs Horizon for the 3 instances(as average of 50 seeds) for the 2 algorithms:





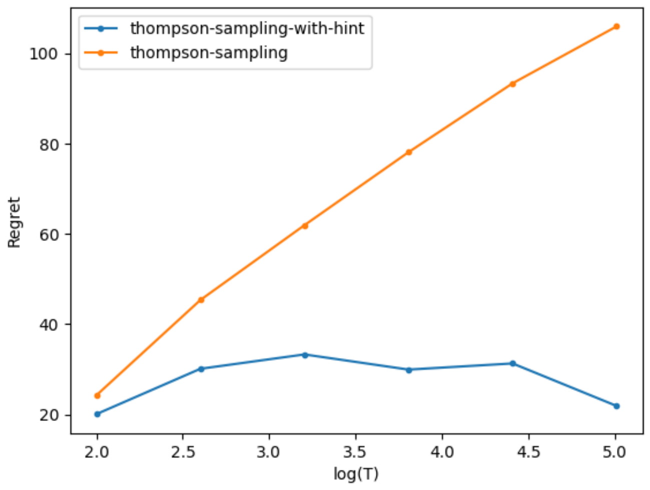


Figure 2: Top: Regret values for Instance i1.txt, Middle: Regret values for Instance i2.txt, Bottom: Regret values for Instance i3.txt. Note: The points are connected by straight lines(linear interpolation)

7.2.2 Observations

- We can clearly observe Thomson sampling with Hint clearly outperforms Thompson Sampling in all 3 instances for all Horizon values.
- The Thompson sampling with hint algorithm gets better and achieves lesser regret sometimes as horizon increases.
- In comparison to straight line curve of Thompson sampling, Thompson sampling with hint gets more or less flat curve specially for higher bandits i.e. i2 and i3 here.
- The inference is that, the Thompson sampling with hint algorithm converges to the most optimal arm after some few horizons(the belief for true mean becomes 1), after which it simply pulls the, most optimal arm always(with high probability), hence the regret doesn't increase. In the graphs they have decreased sometime, this is because this is just the average of 50 seeds, and its high possibility for getting negative regret many times.

7.3 Analysis of T3

Instance	$\epsilon 1$	$\epsilon 2$	$\epsilon 3$
i1	0.00005	0.02	0.5
i2	0.005	0.02	0.5
i3	0.00005	0.02	0.5

Instance	Regret $\epsilon 1$	Regret $\epsilon 2$	Regret $\epsilon 3$
i1	2906.56	463.3	10256
i2	767.34	674.14	10243.82
i3	2294.82	1397	21995.86

Table 1: Left: Epsilon values for 3 instances such that $\epsilon 1 < \epsilon 2 < \epsilon 3$ for each instance, Right: Regret corresponding to those epsilon values such that Regret for $\epsilon 2$ is least for each instance

7.3.1 Observations

- In general for very low and very high epsilon values, the regret is very high.
- For our case, $\epsilon = 0.02$ is optimal value for all 3 instances between the other 2 epsilon values.
- Epsilon values like $\epsilon = 0.5$ is inferred very high, which incurs huge regret.
- Epsilon values like $\epsilon = 0.00005$ is inferred very low, which also incurs huge regret.
- Very high epsilon values like $\epsilon = 0.5$ would mean, the algorithm would explore too much, this leads to many sub optimal arms picking up. Even after knowing most optimal arm by empirical mean, it will still explore with high probability, pulling sub optimal arms and incurring high regret.
- Very low epsilon values like $\epsilon = 0.00005$ would mean, the algorithm would exploit too much, in this case it is very high possibility that algorithm might get stuck in some sub-optimal arm, and is continuously getting exploited, leading to high regrets.