TP2 for Reinforcement Learning

Leman FENG
Email: flm8620@gmail.com
Website: lemanfeng.com

November 30, 2017

Q1

I choose two problems like this, first one contains 7 Bernoulli arms with

$$p = [0.1, 0.1, 0.2, 0.2, 0.3, 0.4, 0.9]$$

second one contains 7 Bernoulli arms with

$$p = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7]$$

Choose horizon $T = 10000$ and repeat each problem for $n = 100$ times then take average. For UCB1 algorithm, we take $\rho_t = 0.2$.

We plot the graph of $\mathbb{E}[R_t]/log(t)$ so it's easy to compare with complexity C_p. Two plots are shown in Figure 1 and 2.

We can see TS algorithm performs better then UCB1. It seems regret of TS is lower than oracle in two figure. Maybe the horizon T is not large enough or average times n is not large enough.

UCB performs so badly because it stuck in a local maximum by choosing a sub optimal arm.

Q2 Non-parametric bandits

For non-parametric bandits, the generalized TS algorithm proposed in the paper says, instead of using the binary reward given by arm to update parameter of beta distribution, we receive a reward $r_t \in [0, 1]$, then we take a binary sample r from Bernoulli distribution with $p = r_t$, then use this sample r to update Beta distribution.

I choose 7 arm of Beta distribution with parameter:

$$((\alpha, \beta)) = [(2, 2), (2, 5), (1, 3), (0.5, 0.5), (5, 2), (3, 4), (1, 1)]$$

The complexity C_p still make sense, we can find in paper [Lai and Robbins], the bound is given in a general form with Kullback-Leibler divergence of two

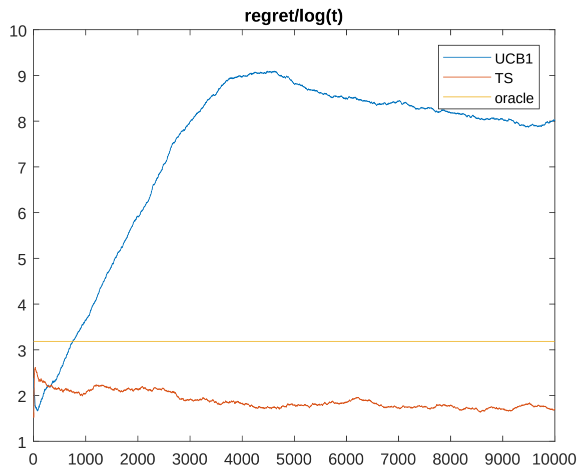

Figure 1: problem 1

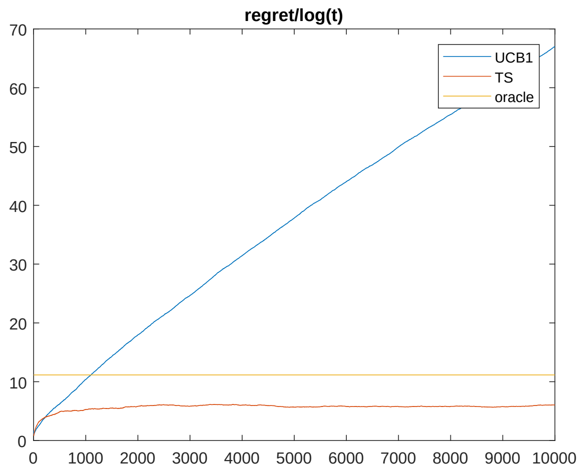

Figure 2: problem 2

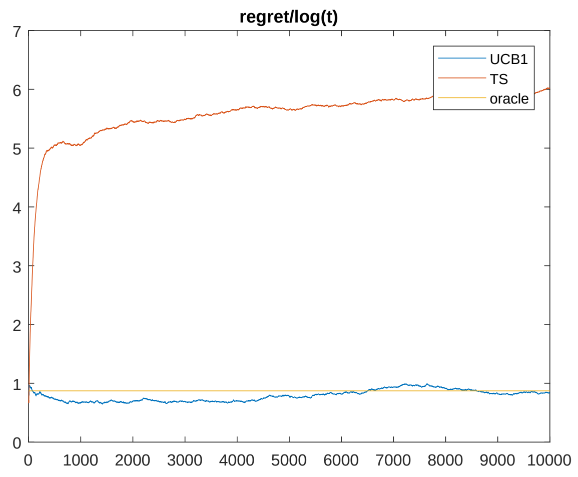

Figure 3: Non parametric

distribution. We have a close form of KL between two beta distributions. So I calculated C_p and plotted it in Figure 3.

This time UCB1 has a better performance and we observed that UCB1 is near the oracle bound.

Q3

I tested 3 algorithms for Movielens data: LinUCB, Random and LinUCB with epsilon greedy.

I take $\alpha = 1$ and $\lambda = 1$ for LinUCB, and $\epsilon = 0.01$. Result is shown in Figure 4. I perform each test for $n = 100$ times then take the average.

θ in LinUCB converge earlier. While LinUCB with epsilon greedy has a better estimate on θ, but it has larger regret because it explored too much to get a unnecessary precision for θ.

Then I played on two parameter α and λ in LinUCB (Figure 5). There is clearly a dilemma between exploration and exploitation.

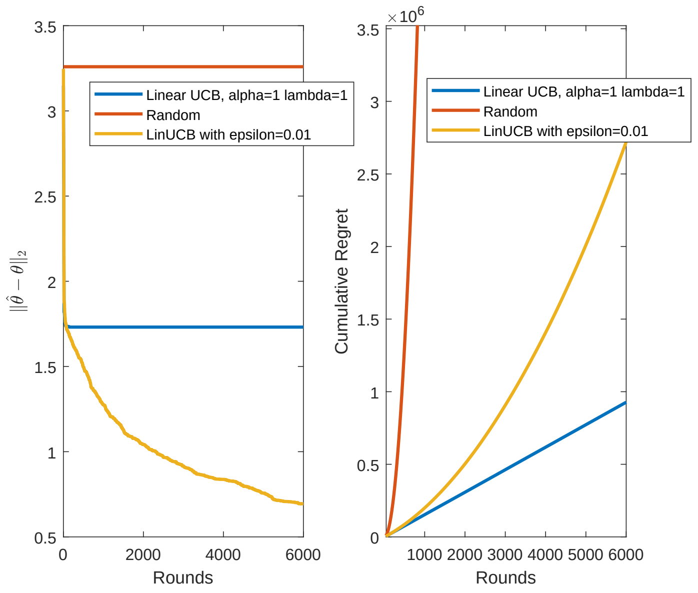

Figure 4: Movielens

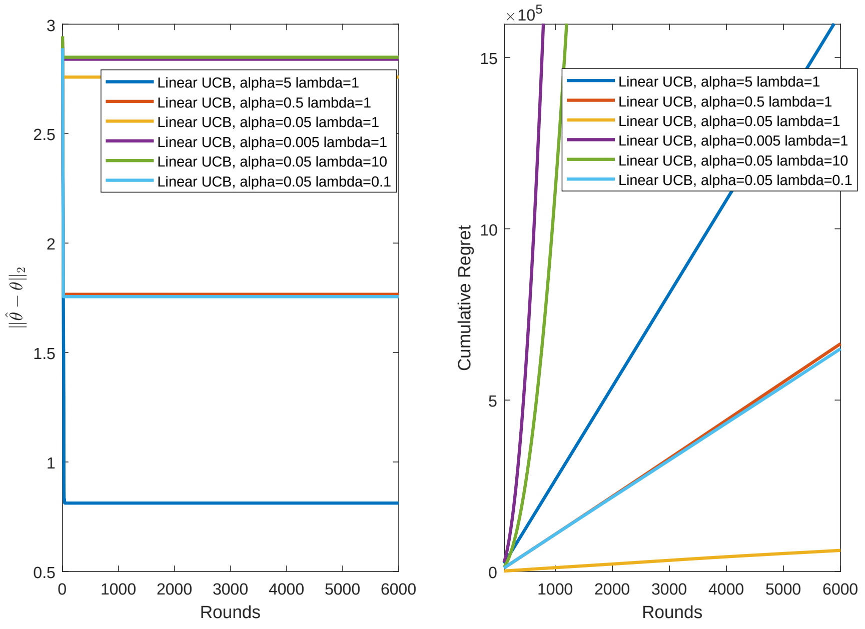

Figure 5: Movielens with different parameters