Reinforcement Learning MVA 2018/2019 TP2

Louis GUO

Exercise 1: Stochastic Multi-Armed Bandits on Simulated Data

Question 1

The Thompson Sampling and the UCB1 are implemented in the class UCB in script UCB.py. Method 'run' can be ran in mode 'UCB1' or 'TS'. A use-example is showed in Part1_MAB1 jupyter notebook "TP2_UCB".

We consider two MAB Bernoulli problems:

- The Bernoulli distributions of the First Problem have means close to one another. Learning on this set should be slowed due to the fact that it is hard to distinguish one arm from another with the variance of each arm.
- The means in the second problem are more separated. Learning on this set should be easier as it is easier to distinguish one arm.

This is quantified by complexity: $c_1 \simeq 13.7$ and $c_2 \simeq 2.4$. By plotting the mean regret of Thompson Sampling (Fig 1 and 2), UCB1 and naive strategies, we observe:

- In both problems, we have linear increasing of the mean regret (in $O(T)$) of naive strategy and the logarithmic increasing in the number of rounds is more visible for Thompson sampling compared to UBC1.
- The UCB1 and Thompson sampling performs better in the easier problem as the algorithms manages to find the optimal arm quickly. For naive, it under-performs in the easier problem probability because the variance of the reward is higher in the harder problem, so is the regret.

The Lai and Robbins lower bound is asymptotic, but for naive and UCB1 algorithms, the lower bound is effective for $T < 8000$. By increasing T, this should be also the case for Thompson sampling algorithm.

Question 2

We consider a MAB problem with Bernoulli, Beta, Finite and Exponential arms. (Fig 3)

For non-Bernoulli bounded arms, UCB1 algorithm can still be used. However, for Thompson sampling, for the posterior distribution to follow a Beta distribution with parameters (Number of successes, Number of fails), the underlying reward process must be of values in $\{0, 1\}$ ie a Bernoulli distribution.

A solution is to force the Bernoulli distribution for non-Bernoulli arms, when a reward $r \in [0, 1]$ is received, we simulate a Bernoulli arm of probability r. This is implemented with the method 'sample'. The complexity seen as lower bound for mean regret still has a meaning but should not be computed with the formula given as it was only for Bernoulli arms (where Kullback-Leibler distance between two Bernoulli distributions can be expressed in terms of their means). We can define more generally the complexity of the inverse of the Kullback-Leibler between the distribution of each arm and the optimal arm.

The problem is considerably harder compared to only-Bernoulli arms, as we can see in the regret curves.

Exercise 2: Linear Bandit on Real Data

The algorithms are implemented in the class linMAB in script linMAB.py. Method 'run' can be ran in mode 'linUCB', 'random' or 'eps_greedy'. A use-example is showed in Part2_linUCB jupyter notebook "TP2_LinUCB".

Question 3

In LinUCB algorithm, there is a theoretical value for α. However, here we calibrate α using cross-validation to adjust to our need in exploration-exploitation dilemma. As shown in Figure 4 and 5, we plot both metrics L2-norm error $||\theta^* - \theta||_2$ and expected cumulative regret for different α. This is a nice illustration of exploration-exploitation dilemma, for high α, exploration is favored, leading to better estimation of θ^* but higher regret score, conversely when α is lowered.

We notice that we don't need a very precise estimation of θ^* to pull the optimal arm. Taking $\alpha = 2$ is sufficient to leverage a good-enough estimation to choose the right arm. We then implement algorithms ϵ-greedy and random algorithms as well and keep track of both metrics.

As seen in Figure 6 and 7, the random strategy estimation θ converges to θ^* as we keep explore randomly but we don't exploit very well as regret is higher compared to the two other strategies. The ϵ-greedy allows more exploitation (with probability $1-\epsilon$), but even when we found the optimal arm, it keeps explore with probability ϵ. This leads to an increasing mean regret with rounds. The LinUCB algorithm finds a good-enough estimation of θ and when this is done, pulls the optimal arm giving a lower mean regret curve.

Figures

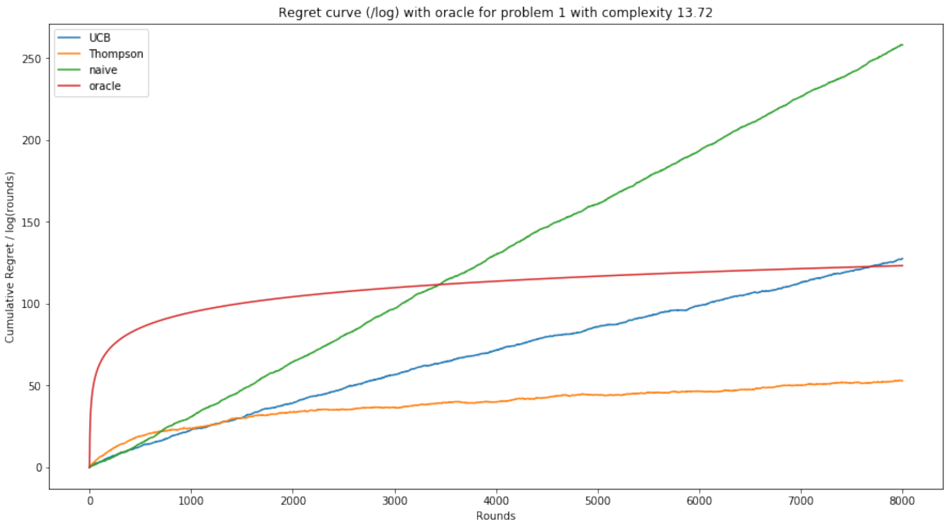

Figure 1: (Q1) Mean regret curve with oracle for (Bernoulli MAB) problem 1.1 with complexity 13.72

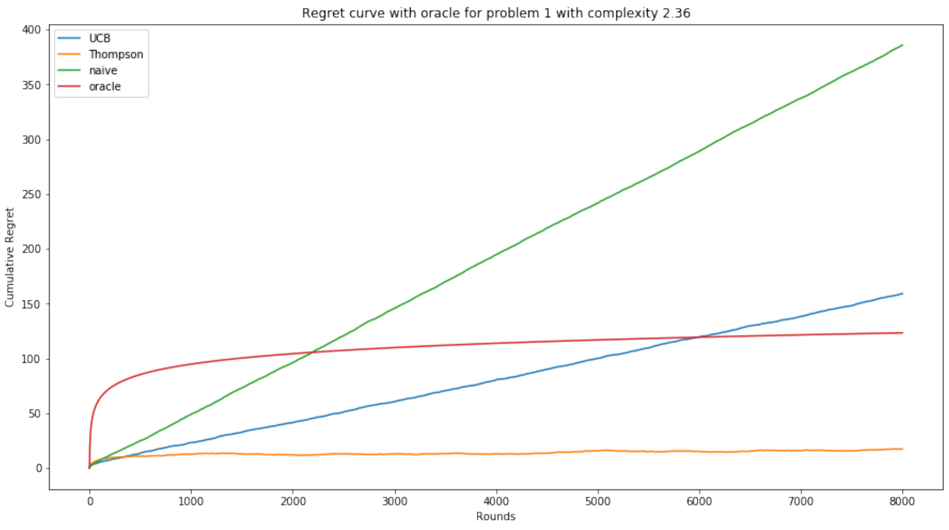

Figure 2: (Q1) Mean regret curve with oracle for (Bernoulli MAB) problem 1.2 with complexity 2.56

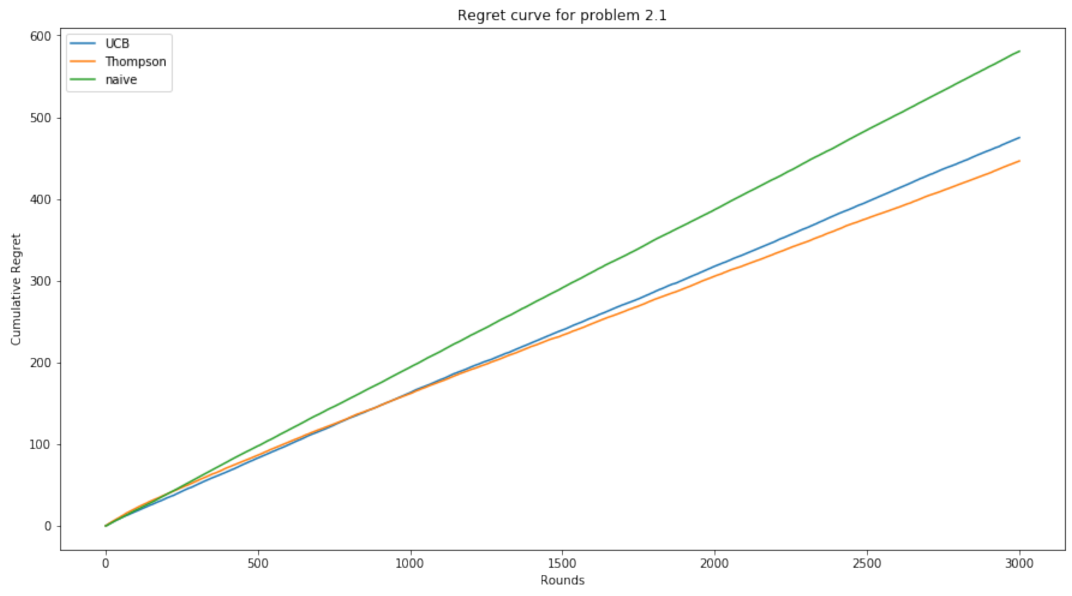

Figure 3: (Q2) Mean regret curve for (not only Bernoulli MAB) problem 2

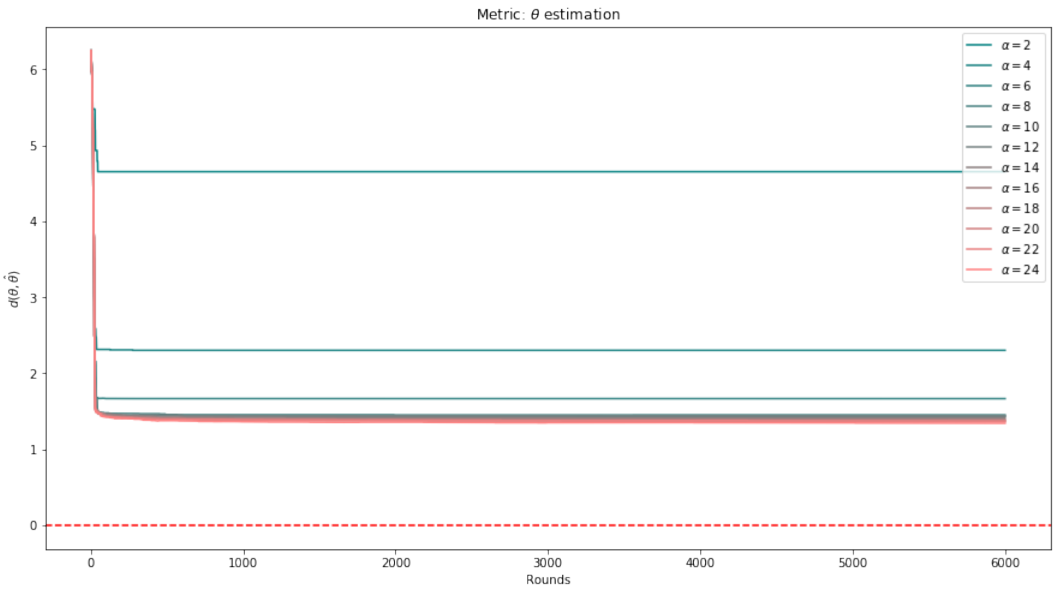

Figure 4: (Q3) $||\theta - \theta^*||_2$ for different values of α

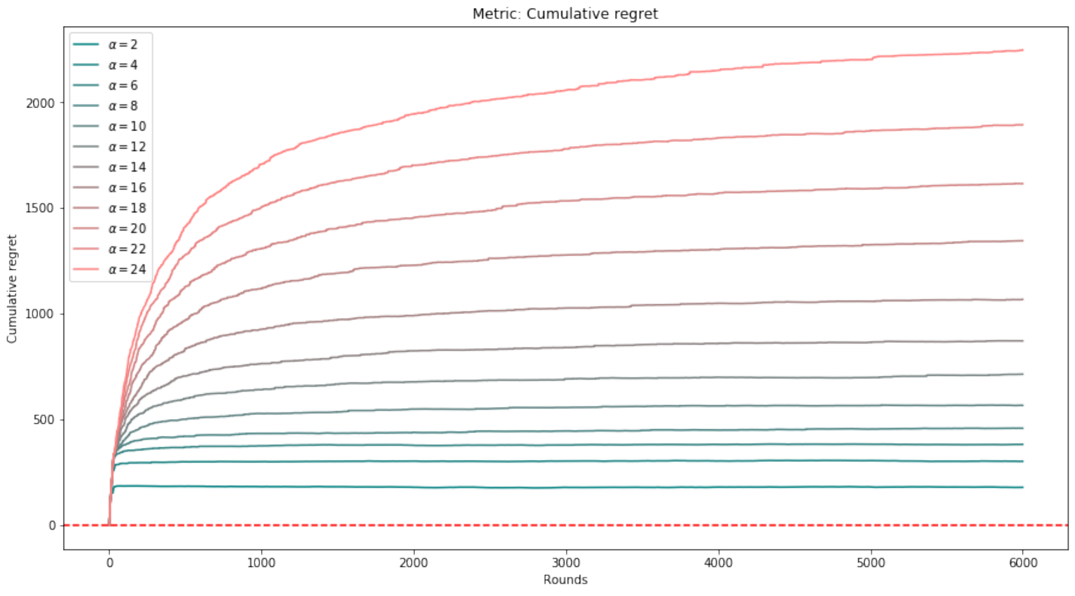

Figure 5: (Q3) Mean cumulative regret for different values of α

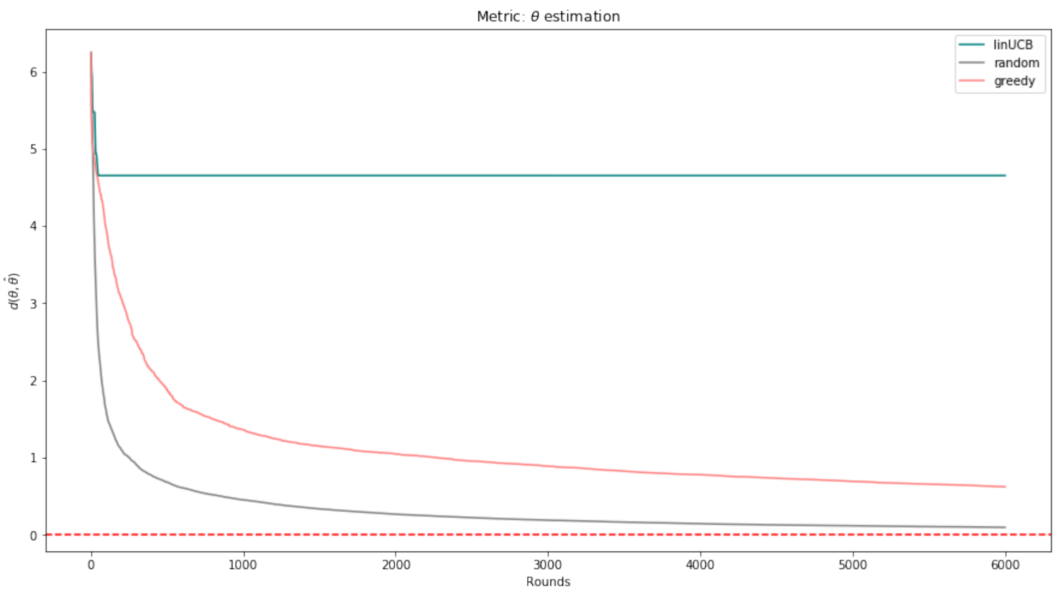

Figure 6: (Q3) $||\theta - \theta^{*}||_2$ for different strategies

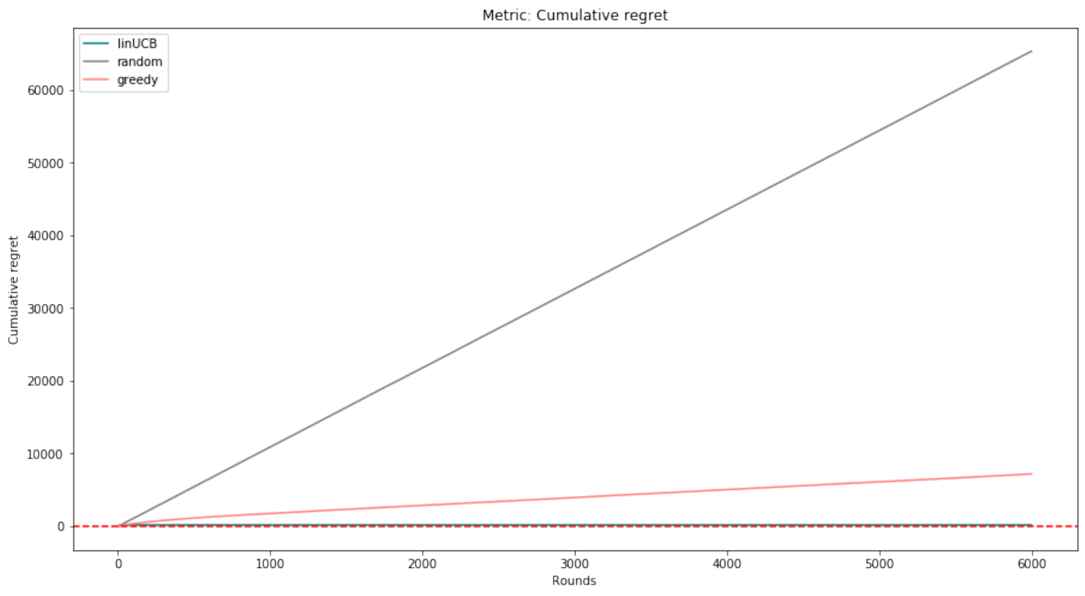

Figure 7: (Q3) Mean cumulative regret for different strategies