EXERCISE 5

Instructor: Yuan Zhou

1 Project Description

In this project, we are going to compare the performance of UCB and LinUCB when the underlying environment is linear bandits. In linear bandits, each arm i is associated with a context vector x_i. And there is also an *unknown* parameter θ such that after each play of arm i, you will get the reward independently generated from distribution $\mathcal{N}(x_i^T\theta, 1)$. Suppose we have n arms denoted by x_i $(0 \le i \le (n-1))$. We assume both of x_i and θ are 3-dimensional vectors. The goal of the policy is to collect as many rewards as possible. We assume the total number of plays is T.

Following from the convention, we will compare the *regret* of different policies. For each policy, the regret is defined as the expected difference between the optimal rewards assuming θ is known and the rewards collected by the policy. Let π denote the policy. Suppose at time t $(1 \le t \le T)$, the choice made by π is i_t and the corresponding collected reward is y_t. To avoid clutter, we have removed the dependency on policy in the notations. Under this scenario, the *regret* is defined as

$$\mathcal{R}_T^\pi \overset{\text{def}}{=} \mathbb{E}\left[T \cdot \max_{0\le i\le(n-1)}\{x_i^T\theta\} - \sum_{t=1}^T y_t\right].$$

From the definition, we know a policy is better then the other one if it incurs a smaller regret.

2 Policies

In this section, we will introduce the policies you are going to implement and compare. From now on, we will use $\bar{\mu}_i(t)$ to denote the empirical average rewards of arm i before the tth time step (exclusive).

2.1 UCB

UCB stands for *Upper Confidence Bound.* During each time step t, instead of playing the arm with the maximum empirical average rewards, it plays the arm with the maximum upper confidence bound. Such bound serves as an optimistic estimate of the arm's real mean rewards. The details are described in Algorithm 1. Note that $T_i(t)$ denotes the number of times arm i is played before time step t (exclusive).

Algorithm 1: UCB

```
1 initialization: play each arm once
2 for time $t = (n+1)$ to $T$ do
3     play arm $\text{argmax}_{0\le i\le(n-1)} \left(\bar{\mu}_i(t) + \alpha \cdot \sqrt{\frac{2\ln(t-1)}{T_i(t)}}\right)$
```

You are suggested to set $\alpha = 0.5$.

2.2 LinUCB

Not like UCB, LinUCB leverages the structure of linear bandits to construct a more refined confidence bound. Let I_d and $\mathbf{0}_d$ denote the identify matrix with dimension $d \times d$ and 0 vector with dimension d respectively. The details are described in Algorithm 2.

Algorithm 2: LinUCB

```
1 initialization: $A = I_3$, $b = \mathbf{0}_3$
2 for time $t = 1$ to $T$ do
3     $\widehat{\theta}_t = A^{-1}b$
4     play arm $i_t = \text{argmax}_{0\le i\le(n-1)} \left(x_i^T\widehat{\theta}_t + \alpha \cdot \sqrt{x_i^T A^{-1} x_i} \cdot \sqrt{\ln(nT^2)}\right)$ and observe
        reward $y_t$
5     update $A = A + x_{i_t}x_{i_t}^T$ and $b = b + x_{i_t}y_t$
```

You are suggested to set $\alpha = 0.1$.

3 Experiment Setup

In the experiment, we will use $T = 1000$ and $\theta = [1, 0, 0]^T$. Let the *empirical regret* be

$$R = T \cdot \max_{0\le i\le(n-1)} \{x_i^T\theta\} - \sum_{t=1}^{T} y_t.$$

For each policy, you will need to run the experiment for 50 trials and report the average empirical regret which is

$$\frac{R_1 + \cdots + R_{50}}{50},$$

where R_k is the empirical regret during the kth trial as the approximation of the real regret.

You will need to produce three figures. Choose $n = 10, 30, 50$ and generate the corresponding figure described as the following.

Minimax Regret Uniformly generate 10 groups of $\{x_i\}_{0\leq i\leq(n-1)}$ from a unit 2-sphere. Plot the maximum average empirical regret among all the generated inputs at time $50, 100, \ldots, 1000$ for every policy.

For the Python language, we have implemented the Uniform Sampling algorithm, which just samples each arm in a round-robin way and generated the figure for data-dependent regret (see Figure 1). To implement a new policy, you only need to inherit the **Learner** class and change the corresponding part of the **main** method. For more details, please check the **code** folder. We have also implemented **sphere_sampling** method in **utils.py** file which you can utilize to generate unifrom samples from a sphere.

If you decide to use another programming language, you need to start from scratch.

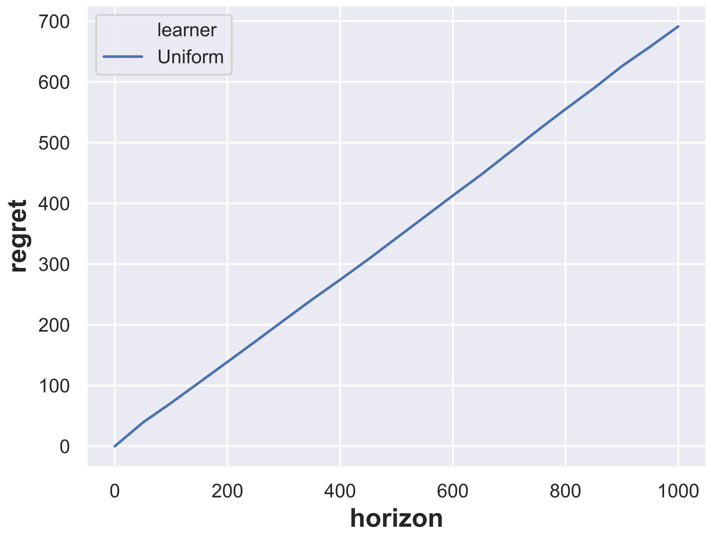

Figure 1: Uniform Sampling

4 Submitting Your Solution

1. Please write a short report with the requested plots and some discussion and put this into a PDF file.

2. Please write your code clearly so that someone else reading the code can understand it without significant effort, i.e., structure it and include enough documentation.

3. Please write a README file that explains how the code should be executed.

Finally pack all files included in items (1)-(3) into a zip archive and upload the archive.