Problem 6 (Programming: Multi-Armed Bandits) (16+10=26 points)

As described in Lecture 17, multi-armed bandits (MAB) a classic formulation for capturing exploration and exploitation trade-off in online machine learning. Let's briefly describe the MAB setting as follows:

- Consider the stochastic K-armed bandit problem, where each arm i is characterized by its reward distribution $\mathcal{D}_i$ with mean θ_i.
- At each time $t = 1, \cdots, T$, the decision maker chooses an arm denoted by $\pi_t \in \{1, \cdots, K\}$ and observes the corresponding random reward X_t, which is independently drawn from the distribution $\mathcal{D}_{\pi_t}$.
- Let $N_i(t)$ and $S_i(t)$ be the total number of plays of arm i and the total reward collected from pulling arm i up to time t, respectively. We define $p_i(t) := S_i(t)/N_i(t)$ as the empirical mean reward up to t.
- Based on the multi-armed bandit convention, our objective is to minimize the *regret* defined as

$$\mathcal{R}(T) := T \cdot \max_{i \in \{1,\cdots,K\}} \theta_i - \mathbb{E}\Big[\sum_{t=1}^{T} X_t\Big],$$

where the expectation is taken with respect to the randomness of the rewards and the employed strategy of the decision maker.

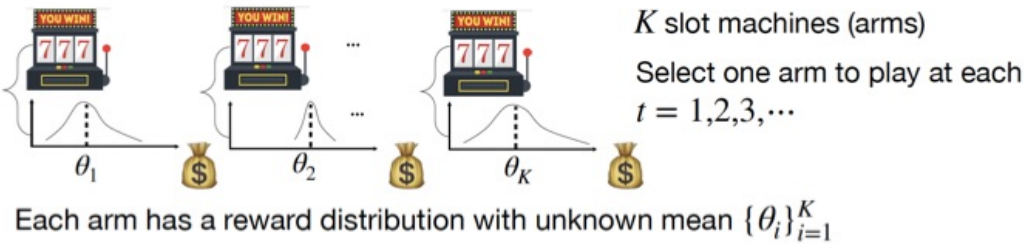

Figure 1: An illustration of the standard MAB problems.

To build an MAB environment as described above, we leverage the popular MAB package **SMPyBandits** (available at `https://github.com/SMPyBandits/SMPyBandits` and can be installed via pip).

In this homework problem, you will implement one classic and useful algorithm called *Epsilon-Greedy* algorithm and compare it with a naive *Empirical Means* algorithm (also known as the *Greedy* algorithm). Under the Epsilon-Greedy method, at each time t, the decision maker chooses an exploration parameter $\epsilon \in [0, 1]$ and enforces some exploration by:

- The decision maker first creates a Bernoulli random variable Z_t with success probability $1 - \epsilon$.
- (Exploitation) If $Z_t = 1$, then the decision maker chooses the arm with the largest empirical mean at time $t - 1$, i.e., $p_i(t - 1)$.
- (Exploration) If $Z_t = 0$, then the decision maker simply selects one of the K arms uniformly at random.

You will do this Python programming task on Jupyter Notebook as in HW2. Please take a look at the notebook "MAB.ipynb" and finish the tasks therein (normally you need no more than 20 lines of code in total).

(a) Please finish the remaining parts of "MAB.ipynb". What are the mean regrets of your Epsilon Greedy algorithm under the three MAB problem instances provided in "MAB.ipynb" and under $\epsilon = 0.01, 0.03, 0.1, 0.3$? Please also compare the performance of Epsilon Greedy algorithm with that of the EmpiricalMeans algorithm.

(b) Suppose we use a diminishing exploration rate in the Epsilon Greedy algorithm , i.e., $\epsilon(t) = t^{-\alpha}$ with $\alpha > 0$? Then, what are the mean regrets under $\alpha = 0.1, 0.5, 1.0$, and 2.0? Can you point out some interesting things from your experimental results?

(a) mean regrets.

	Environment #1 (easy)	Environment #2 (3 groups)	Environment #3 (hard)
$\varepsilon=0.01$	7643.8	5748.2	866.43
$\varepsilon=0.03$	7288.4	3441.9	852.06
$\varepsilon=0.1$	5944.5	1693.1	778.56
$\varepsilon=0.3$	3751.3	1974	615.63
Empirical Means	367.39	75.48	423
UCB	327.68	233.56	486.21
ε-greedy ($\varepsilon=0.1$)	5477	1503.2	754.93

ε-greedy algorithm's regrets are amazingly high (but maybe just my implementation is wrong). 但差距在hard problem減小了。ε對於此algorithm很關鍵, 因為ε過小, 就等同於隨時都在exploitation, 除非很幸運在一開始就有好的arm, 不然基本上應該都在亂選。

(b) ($\varepsilon = t^{-\alpha}$)

	Environment #1 (easy)	Environment #2 (3 groups)	Environment #3 (hard)
$\alpha=0.1$	2831.3	2706.2	476.3
$\alpha=0.5$	3592.2	2104.3	688.42
$\alpha=1.0$	3658.7	1981.1	828.94
$\alpha=2.0$	5057.6	4520.2	877.98

從實驗結果來看，α 太大，太快 decay 會造成沒有足夠多時間去 exploration，導致 regret 過大。