A Comparison of Multi-Armed Bandit Algorithms

CS747: Foundations of Intelligent and Learning Agents
Programming Assignment 01

Report submitted by: Aaron John Sabu

1 Introduction

The assignment deals with the implementation and comparison of different algorithms for sampling the arms of a stochastic multi-armed bandit. Each arm provides iid rewards from a Bernoulli distribution (mean in $(0, 1)$). The objective is to minimise the regret. The implemented algorithms are epsilon-greedy exploration, UCB, KL-UCB, Thompson Sampling, and a variation of the last one given a permutation of the true means. The assignment puts into practice the sampling algorithms discussed in class.

2 Tasks

Task 1: Implementation of the basic algorithms

The basic algorithms - epsilon-greedy, UCB, KL-UCB, and Thompson sampling - have been evaluated with 50 random seeds over horizons of 100, 400, 1600, 6400, 25600 and 102400 steps. All algorithms have been initialized with the random seed as soon as they begin. *bandit.py* contains the implementation of the algorithms written in Python 3.8.2.
The epsilon-greedy algorithm does not begin with pre-initialization of arms. As a result, *np.mean()* may print a RuntimeWarning which is avoided by ignoring via *warnings.filterwarnings*. The implementation of the algorithm is straightforward with an arm being selected either by exploration or exploitation, and a Bernoulli output obtained by picking the arm.
The UCB and KL-UCB algorithms have been merged into a single function which takes an extra input inside the program for determining the type of algorithm. While UCB is easily calculated using mathematical functions, KL-UCB is calculated using binary search with a tolerance of $1.0e^{-4}$. The arm with highest UCB (or KL-UCB) is picked and a Bernoulli output is obtained by picking it. These values are stored in corresponding lists along with other parameters that help determine the total regret of the algorithm. Unlike the epsilon-greedy algorithm, all arms are picked once before the algorithm begins.
Thompson sampling, similar to the epsilon-greedy algorithm, does not involve pre-initialization of arms. Total number of picks, successes, failures and empirical means are stored along with the sequence of picks in order to be utilized in various operations. *numpy.random.beta* is used to provide a value based on the number of successes and failures undergone by the arm. Thompson sampling with hint builds upon the idea of Thompson sampling and is explained in the next subsection.

Task 2: Using a hint in the Thompson sampling algorithm

The problem provides a permutation (sorted by value in this place) to the algorithm. For the first $0.2 \cdot$ Horizon steps, the algorithm follows Thompson sampling. Beyond this period, the algorithm chooses to follow Thompson sampling with probability 0.3, and otherwise, it follows the following algorithm to choose the 'best' arm for that step:

1. Determine the largest true mean from the permutation of means. Let us call it *trueMax*
2. Pick a number *distr* from a probabilistic distribution which peaks at *trueMax* and dies down quickly. The measure of 'quickness' is represented by *phi*. The Beta distribution is chosen with $\alpha = \text{trueMax} \cdot \text{phi}$ and $\beta = (1 - \text{trueMax}) \cdot \text{phi}$ as per the quoted source from Cross Validated (Statistics StackExchange)
3. Pick the arm whose empirical mean is closest to *distr*.

The two parameters - 0.2 and 0.3 - have been chosen based on several iterations using different values, hence setting these in order to obtain a more distinguishable and significant decrease in the regret. Moreover, to match the requirements of the problem, *phi* was suitably placed at $1.0e^{+4}$. This may also be replaced by a value dependent on *trueMax* or some other parameter such as the distance between *trueMax* and the next highest true mean. Special care has to be taken to set *phi* such that both α and β are greater than 1 since otherwise the distribution will flip leading to minimum chance of being at *trueMax*.

Task 3: Find a close-to-optimal ϵ for the epsilon-greedy algorithm

The basic concept of the task is to find a value for ϵ such that a blend of exploration and exploitation is performed rather than overusing either of them. Over a horizon of 102400 steps, several values of ϵ were evaluated. We have $\epsilon_1 < \epsilon_2 < \epsilon_3$, where a suitable evaluation gave $\epsilon_1 = 0.0001$ (too much exploitation), $\epsilon_2 = 0.02$ (a better mix of exploration and exploitation), and $\epsilon_3 = 0.1$ (too much exploration). ϵ_2 gave lower regret for all three instances when compared to ϵ_1 and ϵ_2. The exact values of average regret are given below:

ϵ	I-1	I-2	I-3
0.0001	11235.74	8951.26	49446.88
0.001	1060.26	3657.6	40304.52
0.02	439.4	758.18	44912.26
0.05	1039.26	1126.32	45371.06
0.1	2050.88	2097.42	47291.24
0.8	16399.62	16349.96	43658.92
0.9	18447.88	18408.64	42975.82

Table 1: Average values of regret for various ϵ in the epsilon-greedy algorithm

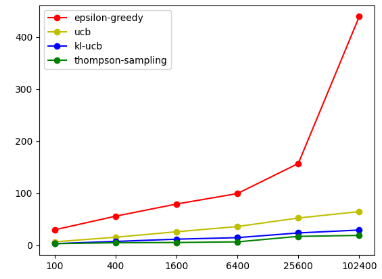
As is clearly visibly, the regret drops on drawing closer to some optimal ϵ and rises after that. The optimal value seems to be close to 0.02. It can also be observed that the above-mentioned value $\epsilon_2 = 0.02$ is closer to this optimal mean than 0.0001 and 0.8 or 0.9. There is a slight reduction in regret for the latter two; however, this is probably due to randomness in picking arms and the corresponding Bernoulli-distributed output since the percent shift is not significantly large and this is not observed in other instances.

Task 4: Comparison of the four basic algorithms

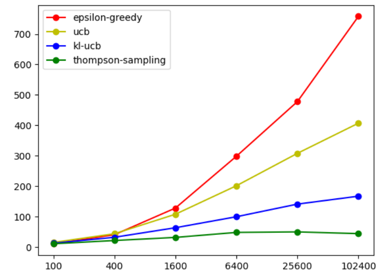
This task involved plotting the curves of each algorithm over several values of horizons covering all three instances. These are given below for each corresponding task.

Task 1

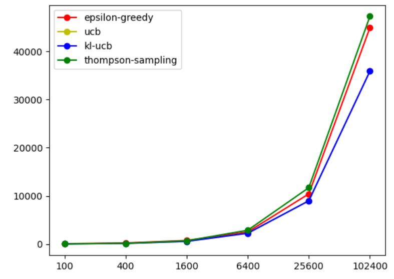
We obtain the following plots for the three instances:



(a) Instance 1



(b) Instance 2



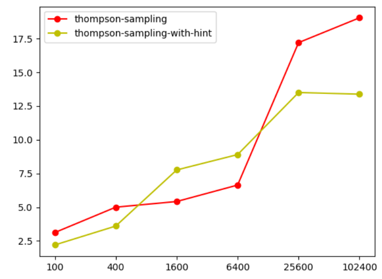
(c) Instance 3

Figure 1: Task 4.1

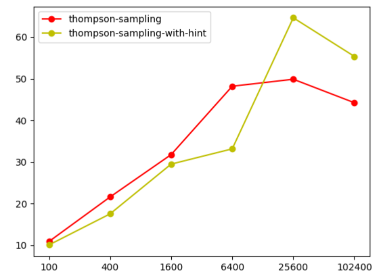
Thompson sampling, KL-UCB and UCB algorithms (in the order) are superior in obtaining smaller regrets with respect to the epsilon-greedy algorithm as per figures 1 and 2. It may seem from the third instance that Thompson sampling is not performing well. However, it is visible that the rate of increase of Thompson sampling is less than that of the epsilon-greedy algorithm and that the former will most probably perform better on extrapolating to larger horizons.

Task 2

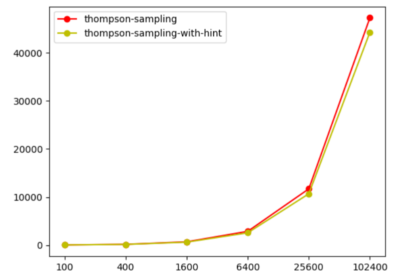
We obtain the following plots for the three instances:



(a) Instance 1



(b) Instance 2



(c) Instance 3

Figure 2: Task 4.2

It is clear from figures 1 and 3 that, when given a hint, Thompson sampling can perform with smaller regret. Figure 2 seems to give an opposite impression about this result. However, it should be noticed that the reason behind the increase from 6400 to 25600 should mostly be related to the stochastic picking of arms rather than an inferior quality of algorithm, and the algorithm will mostly perform better with the hint even in this case provided different random seeds or larger horizons.