CS 747: Programming Assignment 1

In this assignment, you will implement and compare different algorithms for sampling the arms of a stochastic multi-armed bandit. Each arm provides i.i.d. rewards from a Bernoulli distribution with mean in (0, 1). The objective is to minimise regret. The algorithms you will implement are round-robin sampling, epsilon-greedy exploration, UCB, KL-UCB, and Thompson Sampling. This is a relatively straightforward assignment, which essentially puts to practice the algorithms we have discussed in class.

Data and Scripts

This directory has (1) a subdirectory called `instances` containing three bandit instances, (2) a `submission` subdirectory in which you will place all your submission-related material. (3) a script `check.sh` inside `submission` for verifying the input-output behaviour of your program, and (4) a file `outputFormat.txt` to illustrate the format of output your code is expected to generate.

Bandit instances are encoded as text files with the mean rewards of the arms provided one to a line. Hence the number of lines gives the number of arms. The instances provided have 2, 5, and 25 arms.

The program you create will have to take in a number of command line arguments. By running the checking script provided, you can make sure that your program consumes the input parameters correctly and produces the desired output.

You will have to perform multiple random executions of different algorithms on the given instances, and place all your data in a text file. The example provided shows you the expected format.

All your submission materials should go into the `submission` folder, which you will compress and submit.

Task

You have to write code, generate data by running the code, and compile graphs and a report based on the data.

Code

You will have to prepare a script called `bandit.sh` that performs the following functions. You may use any programming language of your choice internally to code up these functionalities; you can also decide for yourself how to modularise the code and name the internal functions. What we shall insist upon is the input-output behaviour of `bandit.sh`, which we describe below.

`bandit.sh` must accept the following command line parameters.

> `--instance in`, where `in` is a path to the instance file.
> `--algorithm al`, where `al` is one of `round-robin`, `epsilon-greedy`, `ucb`, `kl-ucb`, and `thompson-sampling`.
> `--randomSeed rs`, where `rs` is a non-negative integer.
> `--epsilon ep`, where `ep` is a number in [0, 1].
> `--horizon hz`, where `hz` is a non-negative integer.

(Once you have finished coding bandit.sh, run `check.sh` to make sure that you correctly read in all the command line parameters. While testing your code, we will use a different version of `check.sh`—with different parameters—and call it inside your `submission` directory.)

Your first task is to simulate a multi-armed bandit. You must read in the bandit instance and have a function to generate a random 0-1 reward with the corresponding probability when a particular arm is pulled. A single random seed will be passed to your program; you must seed the random number generator in your simulation with this seed. If any of your algorithms are randomised, they must also use the same seed for initialisation.

Given a seed and keeping other input parameters fixed, your entire experiment must be **deterministic**: it should execute the same way and produce the same result. Of course the execution will be different for different random seeds; the point being made is that of repeatability for a given seed. You should be able to implement this property by initialising all the random number generators in your program based on the seed provided as an input: you should not leave them unseeded or use strategies such as seeding based on system time. Make sure you understand this requirement; if the behaviour of your code does not get fixed by the input random seed (keeping other input parameters fixed), you will lose 6 marks.

Having set up the code to pull arms and generate rewards, you must implement the following sampling algorithms: (1) round-robin, (2) epsilon-greedy, (3) UCB, (4) KL-UCB, and (5) Thompson Sampling. You are free to make assumptions on unspecified aspects such as how the first few pulls get made, how ties get broken, how any algorithm-specific parameters are set, and so on. But you must list all such assumptions in your report. The only external parameter to the given set of algorithms is epsilon for epsilon-greedy sampling, which will be passed from the command line. Recall that on every round, an algorithm can only use the sequence of pulls and rewards up to that round (or statistics derived from the sequence) to decide which arm to pull.

Specifically, it is illegal for an algorithm to have access to the bandit instance itself (although `bandit.sh` has such access).

Passed an instance, a random seed, an algorithm, epsilon, and a horizon, your code must run the algorithm on the instance for "horizon" number of pulls and note down the cumulative reward REW. Subtracting REW from the maximum expected cumulative reward possible (the product of the maximum mean and the horizon) will give you REG, the cumulative regret for the particular run. Note that this number can be negative (and might especially turn up so on small horizons—why?). When the algorithm terminates, `bandit.sh` should output a **single** line with six entries, separated by commas and terminated with a newline ('\n') character. The line must be in this format; `outputFormat.txt` contains a few such lines (in which REG is set to arbitrary values just for the purpose of illustration).

> instance, algorithm, random seed, epsilon, horizon, REG

We will run your code on a subset of input parameters and validate the output with an automatic script. You will lose 6 marks if your code does not produce output in the format specified above.

Note that epsilon only needs to be used by `bandit.sh` if the algorithm passed is `epsilon-greedy`; for other algorithms it is a dummy parameter. Your output must still contain epsilon (either the value passed to it or any other value) to retain the six-column format.

Output Data, Plots, Report

Having written `bandit.sh`, run it for every combination of

> **instance** from `"../instances/i-1.txt"`; `"../instances/i-2.txt"`; `"../instances/i-3.txt"`,
> **algorithm** from `round-robin`; `epsilon-greedy` with `epsilon` set to 0.002, 0.02, 0.2; `ucb`, `kl-ucb`, `thompson-sampling`,
> **horizon** from 50; 200; 800; 3200; 12800; 51200; 204800, and
> **random seed** from 0; 1; ...; 49.

It is best that you write your own wrapper script for generating the output for all these input configurations. Place all the output lines in a single file named `outputData.txt`. Notice that the file must have exactly 3 × 7 × 7 × 50 = 7350 lines. It will take you a considerable amount of time to generate data (especially for longer horizons), and so do not leave this task to the last minute. Since data for shorter horizons will anyway be generated as a part of the longer-horizon experiments, you might be able to save some time by recording intermediate regret values. However, your submitted `bandit.sh` script must still only print a single line corresponding to the horizon passed to it.

You will generate three plots: one for each instance. The plot will have horizon on the x axis (use a log scale) and regret on y axis. It will contain 7 lines: one for each algorithm (counting epsilon-greedy as three algorithms). Each point will give the average regret from the fifty random runs at the particular horizon for the algorithm. Make sure you provide a clear key so the plot is easy to follow.

Include all three graphs in a file called `report.pdf`, which should also state any assumptions in your implementation and provide your interpretations of the results. Feel free to put down any observations that struck you while working on this assignment. Do not leave your graphs as separate files: they must be embedded in `report.pdf`.

Submission

Place these items in the `submission` directory.

> `bandit.sh` and all the code that it needs to run.
> `outputData.txt`
> `report.pdf`
> `references.txt` (see the section on Academic Honesty on the course web page)

Compress the directory into `submission.tar.gz` and upload on Moodle under Programming Assignment 1.

Evaluation

We will evaluate you based on your report, and also run your code to validate the results reported. If your code does not run on the sl2 machines or your report is absent/incomplete, you will not receive any marks for this assignment.

1 mark each is reserved for the correctness of your implementation of the round-robin, epsilon-greedy, and UCB algorithms, and 2 marks each for KL-UCB and Thompson Sampling. Your presentation of the results and accompanying observations in your report carry 3 marks.

The TAs and instructor may look at your source code and notes to corroborate the results obtained by your agent, and may also call you to a face-to-face session to explain your code.

Deadline and Rules

Your submission is due by 11.55 p.m., Tuesday, September 3. Finish working on your submission well in advance, keeping enough time to generate your data, compile the results, and upload to Moodle.

Your submission will not be evaluated (and will be given a score of zero) if it is not uploaded to Moodle by the deadline. Do not send your code to the instructor or TAs through any other channel. Requests to evaluate late submissions will not be entertained.

Your submission will receive a score of zero if your code does not execute on the sl2 machines. To make sure you have uploaded the right version, download it and check after submitting (but before the deadline, so you can handle any contingencies before the deadline lapses). If your code needs any special libraries to run on the sl2 machines, it is **your** responsibility to get them installed. You can do so by filing a bug with the CSE system administrators.

You are expected to comply with the rules laid out in the "Academic Honesty" section on the course web page, failing which you are liable to be reported for academic malpractice.