CS 747: Programming Assignment 1

(Prepared by Santhosh)

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 the regret. The algorithms you will implement are epsilon-greedy exploration, UCB, KL-UCB, Thompson Sampling, and some variation of Thompson Sampling. This assignment will essentially put into practice the sampling algorithms 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, (4) a script `verifyOutput.py` inside `submission` for verifying that your submission data is printed in correct format and reproducible, and (5) 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 several 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.

Tasks

In this assignment, you should do the following tasks.

T1) Implement the sampling algorithms: (1) epsilon-greedy, (2) UCB, (3) KL-UCB, and (4) Thompson Sampling. (This task is straightforward, as discussed in class.)

T2) This task requires a little thought. In this task, we give you some knowledge of the true means of the arms. Can you do better than the regular sampling algorithms? More precisely, we give you the true means of the arms without revealing their arm numbers (say we provide a random permutation of the true means). If you think that you can do better given this knowledge, implement an algorithm based on the principles of Thompson sampling. The idea is that using the extra knowledge, your algorithm must perform better than regular Thompson sampling. Let us call this new algorithm as `(5) thompson-sampling-with-hint`. It is a good idea to to avoid having tunable parameters in your algorithm. If you use such parameters, you will have to run with the same static values for all bandit instances (cannot tune based on the instance).

T3) This is fairly a simple experiment. You need to play with different values of epsilon for epsilon-greedy. Specifically, is it possible to give three epsilon values $\varepsilon 1 < \varepsilon 2 < \varepsilon 3$ for each instance such the regret of $\varepsilon 2$ is less compared to the other two? If possible, give those three ε values for each instance.

T4) Generate data by running the code written in the above tasks, and compile graphs and a report based on the data.

These tasks are elaborated in the following subsections.

T1 & T2: Coding

This time we are not giving you the liberty of writing code in any programming language of your choice. You must code in Python 3.8.2 and not use any special libraries except Numpy v1.19 (to work with vectors and matrices) and Matplotlib (for generating plots).

You will have to prepare a source file called `bandit.py` that performs the following functions. You can 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.py`, which we describe below.

`bandit.py` 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 `epsilon-greedy`, `ucb`, `kl-ucb`, `thompson-sampling`, and `thompson-sampling-with-hint`.
- `--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.

Your first job 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 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 not receive any marks for the assignment.

Having set up the code to pull arms and generate rewards, you must implement the following sampling algorithms: (1) epsilon-greedy, (2) UCB, (3) KL-UCB, (4) Thompson Sampling, and (5) Thompson Sampling with Hint. 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.py` has such access.

To give the permutation of the true means for `thompson-sampling-with-hint`, sort the true means and pass it as an argument to `thompson-sampling-with-hint`. That is, use a line of code similar to the following to generate hint.
`hint_ls = numpy.sort(true_mean_ls)`

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.py` 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 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 not receive any marks for the assignment if your code does not produce output in the format specified above.

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

Note that epsilon only needs to be used by `bandit.py` 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.

T3: Experiments on epsilon-greedy

For the experiments in T3, take the average REG of 50 random seeds (0 through 49) for the horizon of 102400.

T4: Output Data, Plots, Report

Having written `bandit.py`, now it's time to generate some data and compare their performances. Generate data for the code written in T1 and T2 separately.

To generate data for T1, run `bandit.py` for every combination of

> **instance** from `"../instances/i-1.txt"`; `"../instances/i-2.txt"`; `"../instances/i-3.txt"`,
> **algorithm** from `epsilon-greedy` with `epsilon` set to 0.02; `ucb`, `kl-ucb`, `thompson-sampling`,
> **horizon** from 100; 400; 1600; 6400; 25600; 102400, and
> **random seed** from 0; 1; ...; 49.

To generate data for T2, run `bandit.py` for every combination of

> **instance** from `"../instances/i-1.txt"`; `"../instances/i-2.txt"`; `"../instances/i-3.txt"`,
> **algorithm** from `thompson-sampling`, `thompson-sampling-with-hint`
> **horizon** from 100; 400; 1600; 6400; 25600; 102400, 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 of T1 in a file named `outputDataT1.txt` and similarly place all the output lines of T2 in a file named `outputDataT2.txt`. Notice that the files `outputDataT1.txt` and `outputDataT2.txt` must have exactly $3 \times 4 \times 6 \times 50 = 3600$ and $3 \times 2 \times 6 \times 50 = 1800$ lines respectively. It will take you at least 5 to 10 hours (depending on your system configuration) 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.py` file must still only print a single line corresponding to the horizon passed to it.

You will generate six plots: two for each instance (T1 & T2 separately). The plot will have horizon on the x axis (use a log scale) and regret on y axis. The plot for T1 will contain 4 lines: one for each algorithm and the plot for T2 will contain two lines: `thompson-sampling` and `thompson-sampling-with-hint`. Each point will give the average regret from the 50 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 six graphs in a file called `report.pdf`, which should also state any assumptions in your implementation and provide your interpretations of the results. Also briefly explain your algorithm for thompson-sampling-with-hint. 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.py` and all the code that it needs to run.
- `outputDataT1.txt`
- `outputDataT2.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 it 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 cs747 docker container or your report is absent/incomplete, you will not receive any marks for this assignment.

5 marks reserved for correctness of your implementation in T1 and, the report containing the plots and interpretation of the results of the algorithms in T1. (1 mark each for epsilon-greedy and UCB, and 1 1/2 each for KL-UCB and Thompson Sampling. Please note: Absence of the plots & implementation details of the algorithm will lead to zero marks for that particular algorithm.)

4 marks for T2. We will evaluate based on its performance against regular Thompson Sampling, and the report which contains the plots and brief description of your algorithm and your observations.

1 mark for T3. Give your response in `report.pdf` . No need to submit any graphs.

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., Friday, September 25. 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 cs747 docker container. 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).

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.