CS 747: Programming Assignment 1

(TAs in charge: Rahul Sharma, Akshay Arora)

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 fixed distribution. The objective is to minimise regret. The algorithms you will implement are 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. The only departure from the class discussion is in the reward distributions of the arms. In class we assumed these distributions to be Bernoulli: that is, only generating 0's and 1's as rewards. In this assignment, the rewards can be *arbitrary* real numbers that lie in the interval [0, 1]. This is not a very significant difference, but one that you must handle in your algorithms anyway.

Code

You will find three directories in this code base.

- The `server` directory comprises the code to simulate a bandit, in other words the "environment". The server waits for a client ("agent") to connect with it and start pulling arms in sequence. For each pull, the server generates a reward in [0, 1] based on the true mean of the arm pulled (known to the server, but unknown to the client), which is communicated back to the agent. Each communication from the server also contains the total number of pulls performed so far. Thus, each server message is of the form "reward,pulls". Agent-server communication happens through a TCP connection. The server is coded in C++, and compiles into a binary called `bandit-environment` on running `make`. It can be started with a shell script called `startserver.sh`.

- The `data` directory contains six bandit instances, each with either 5 or 25 arms, and each with the reward distributions of its arms being `bernoulli`, `histogram`, or `betaDistribution` (all arms in a bandit instance have the same family of distributions). The data provided in the instance files, along with the code that consumes them in the server, should make it clear exactly how stochastic rewards are generated in each of these cases. Do create and experiment with other instances for testing your algorithms.

- The `client` directory is provided to you as an example of what your submission must achieve. The client is the agent that implements sampling algorithms. The agent provided to you merely samples the arms in a round-robin fashion: you will have to implement more efficient sampling algorithms. Observe that the

command line parameters to the client include `--algorithm` (which can take values `epsilon-greedy`, `UCB`, `KL-UCB`, and `Thompson-Sampling` (all case-sensitive)), and `--epsilon` (which will be a real number in [0, 1]). The agent provided is coded in C++: it compiles using `make` into a binary called `bandit-agent`.

Run `startexperiment.sh` in the top level directory to run an experiment. It will start the server and the client in sequence. The server writes out a log file based on its interaction with the client; the per-step reward and regret are available from this log file. Run with different algorithms, random seeds, bandit instances, and horizons to get familiar with the space of experiments.

Submission

You will submit two items: (1) working code for your agent, which implements different algorithms, and (2) a report containing graphs of regret measured at different horizons, as well as your observations from the experiments.

- You must submit a directory called `client`, which contains all the source and executable files of your agent. The directory must contain a script named `startclient.sh`, which must take in all the command line arguments that are accepted by the example client provided. You are free to build upon the provided C++ agent, or otherwise implement an agent in any programming language of your choice. The hostname and port number should suffice for setting up a TCP connection with the server.

  Your code will be tested by running an experiment that calls `startclient.sh` in your directory. Before you submit, make sure you can successfully run `startexperiment.sh` on the `sl2` machines (`sl2-X.cse.iitb.ac.in`, where `X` is the machine number, for example 88).

- For each of the six bandit instances provided in the `data` directory; each algorithm from {`epsilon-greedy`, `UCB`, `KL-UCB`, `Thompson-Sampling`}; each horizon in {1000, 10000}, generate at least 100 runs by varying the random seed. Run at least 5 versions of `epsilon-greedy` on each instance, with different values of `epsilon`, in order to demonstrate the effect of this parameter. Create a separate plot for each bandit instance, with the number of pulls (up to the horizon) on the x axis and the (expected cumulative) regret on the y axis. Plot a curve corresponding to each algorithm (thus, a total of 8 curves). Each point must be the average over all runs for the corresponding setting of instance, horizon, and algorithm. `epsilon-greedy`

  Place your plots in a single pdf file called `report.pdf`, and along with the plots, note down your observations about the results obtained. What patterns do they

display? Do your results validate your intuition about the effect of epsilon? Did you notice anything unexpected? Do you have any ideas to further improve performance?. The report should mention what changes you made (if any) to account for non-Bernoulli rewards. It should also include other relevant details about your implementation of the algorithms (for example, if you tuned any parameters).

Place all the data generated from your experiments, as well as `report.pdf`, within a directory called `report`, and place this directory inside `client`.

If you have used any external code snippets or libraries in your code, make sure you provide references in your report. It is okay to use public code for parts of your agent such as the network communication module, or, say, libraries for random number generation and sorting. However, the logic used for sampling the arms must entirely be code that you have written.

In summary: you must submit your `client` directory (compressed as `client.tar.gz`) through Moodle. The directory must contain `startclient.sh`, along with all the sources and executables for the agent. The `client` directory must also contain a directory called `report`, in which you provide all the data generated by your experiments, and a file called `report.pdf`.

Evaluation

Your algorithms will be tested on different bandit instances and horizons to verify that they perform as expected. Their performance on the two provided bandit instances, as well as your accompanying report, will carry 6 marks. The performance of the algorithms on other unseen bandit instances will carry 4 marks. For each such (algorithm, instance, horizon) configuration, a large number of runs (say 100) will be conducted by varying the random seed passed to the server and agent. The average regret over these runs will be recorded. The number of bandit arms will be between 2 and 50; the horizon will be between 1 and 100,000.

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., Sunday, August 19. Finish working on your submission well in advance, keeping enough time to upload it 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.

Before submission, make sure that your code runs for a variety of experimental conditions. Test your code on the `sl2`

machines even while you are developing it: do not postpone this step to the last minute. If your code requires any special libraries to run, it is *your* responsibility to get those libraries working on the `sl2` machines (go through the CSE bug tracking system to make a request to the system administrators). Make sure that you upload the intended version of your code to Moodle (after uploading, download your submission and test it on the `sl2` machines to make sure it is the correct version). You will not be allowed to alter your code in any way after the submission deadline. In short: your grade will be completely determined by your submission on Moodle at the time of the deadline. Play safe by having it uploaded and tested at least a few hours in advance.

You must work alone on this assignment. Do not share any code (whether yours or code you have found on the Internet) with your classmates. Do not discuss the design of your agent with anybody else. Do not see anybody else's code or report.