CS747 - Assignment 1 Report

Shashank Kumar

September 15, 2020

Directory Structure

- *bandit.py* - Main program file to run a bandit instance
- *run.py* - Wrapper to run the file *bandit.py* to generate outputs
- *plot.py* - Script to plot the generated outputs from *run.py*
- *epsilonGreedy.py* - Code to implement epsilon-greedy algorithm
- *ucb.py* - Implementation of UCB algorithm
- *ucb_kl.py* - KL-UCB algorithm implementation
- *thompson_sampling.py* - Code for regular Thompson sampling
- *thompson_sampling_hint.py* - Thompson sampling with the true means known
- *helper.py* - A file for additional tasks like reading and parsing instances

---

Implementation Details

Epsilon-Greedy

A number is drawn from a uniform distribution in the range [0, 1], say `n`. Based on the epsilon (ϵ) value specified, the arm with maximum empirical mean is chosen if the `n` $\leq (1-\epsilon)$. Otherwise an arm is chosen randomly from the bandit instance.

UCB

The UCB means for all the arms are initialized to be ∞. For each run, the arm with maximum UCB mean is chosen and it's mean gets updated based on it's empirical mean, the time `t`, and number of times it has been sampled u_a^t.

KL-UCB

The KL-UCB equation is as follows:

$$ucb\text{-}kl_a^t \;=\; max\{q\,\epsilon\,[\hat{pa}^t, 1]\ s.t.\ u_a^t * KL(\hat{pa}^t, q) \leq ln(t) + c * ln(ln(t))\ where\ c \geq 3\}$$

For KL-UCB algorithm implementation, the constant c is chosen to be 3.
The amount by which the value of q is decreased after every step if the equation is not satisfied, is taken to be 0.1. Rest is similar to UCB implementation.

Thompson-sampling

Initial number of success and failures for all the arms are taken to be zero. A random number is drawn for each arm from the Beta distribution using numpy as

$$Beta(a, b) = Beta(\#success + 1, \#failures + 1)$$

The arm for which the value drawn is maximum is chosen.

Thompson-sampling with hint

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Generating Rewards

Reward from a chosen arm is based on a fair coin-toss simulated by drawing a number from a uniform distribution (say `n`). Let $\hat{p}$ be the true mean of the chosen arm. If $n < \hat{p}$, a reward of 1 is awarded. This same scheme is followed for all the algorithms.

Miscellaneous

- To display true means, empirical means, and the number of pulls of an arm, run the file *bandit.py* with the argument *--verbose*. By default, verbosity is turned off.
- The arm with the lower arm number is given priority in case of ties.
- The file to which output is written is labelled as *outputData.txt*. This file is generated in the same directory as *bandit.py*.
- Comments have been provided wherever seemed appropriate

Plots

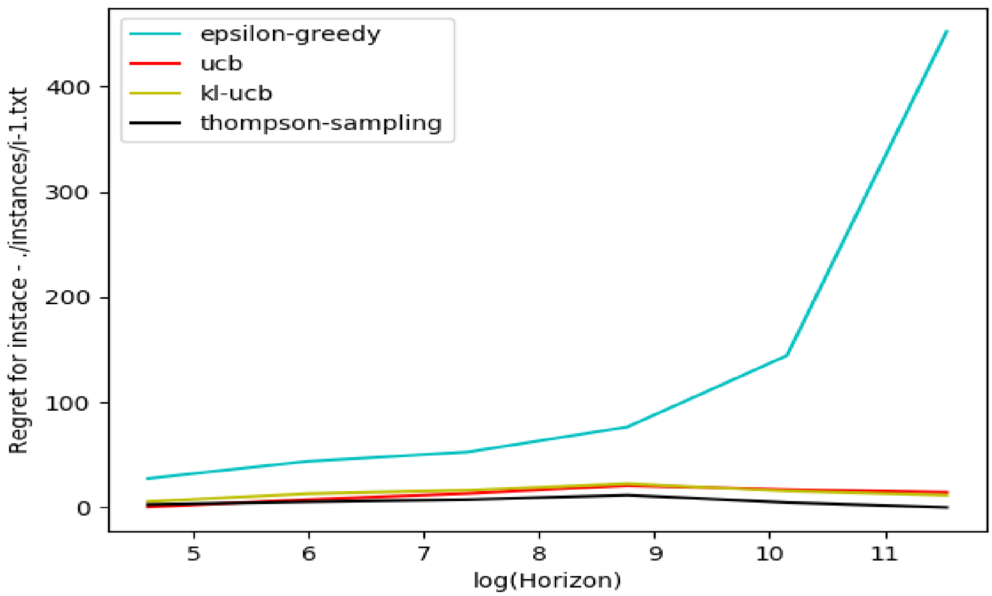

Figure 1: Comparison of regret for the four algorithms on instance i_1

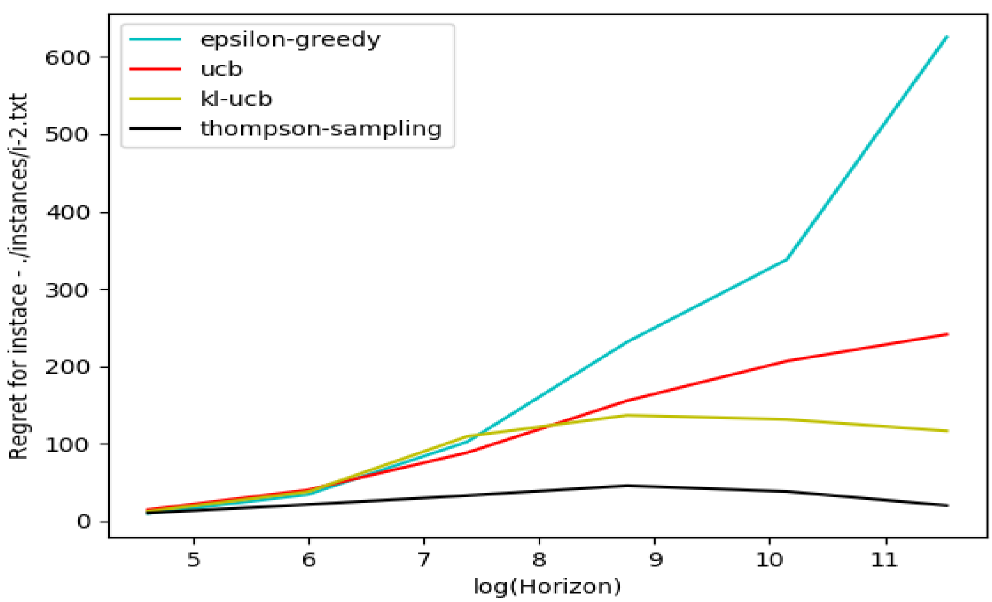

Figure 2: Comparison of regret for the four algorithms on instance i_2

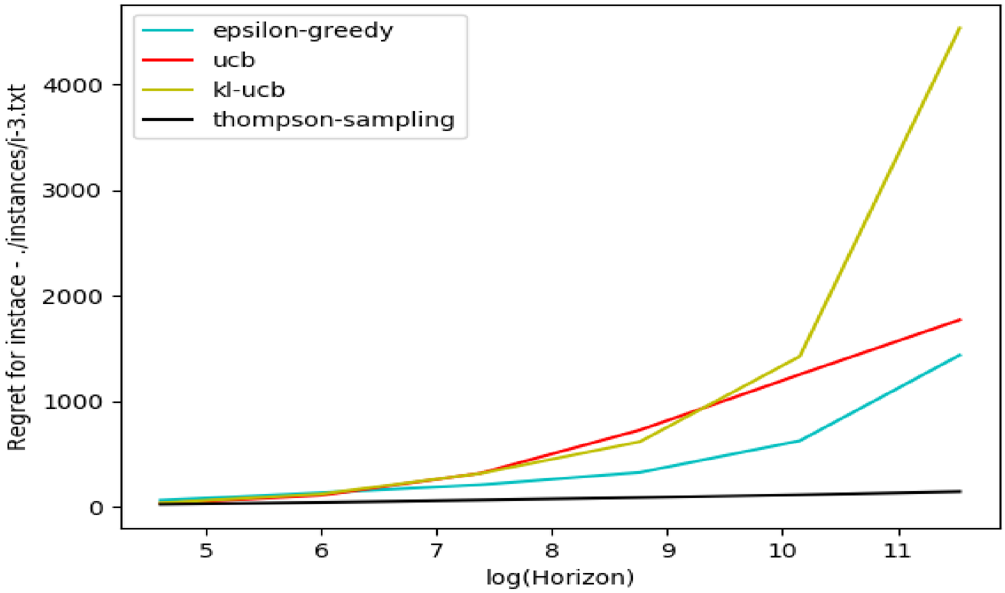

Figure 3: Comparison of regret for the four algorithms on instance i_3

plots/T2-i1.png

Figure 4: Comparison between Thompson-sampling with and without hint on instance i_1

plots/T2-i2.png

Figure 5: Comparisonbetween Thompson-sampling with and without hint on instance i_2

plots/T2-i3.png

Figure 6: Comparison between Thompson-sampling with and without hint on instance i_3

Conclusions

The regret for epsilon-greedy algorithm increases fastest as the horizon increases when compared to other algorithms. Thompson-sampling and KL-UCB, on the other hand, have a much better performance in terms of regret. The regret for UCB, on an average, falls between these two cases. This can be seen with the plots above.

References

[1] Python documentation available at `https://docs.python.org/3/`

[2] Matplotlib documentation available at `https://matplotlib.org/`

[3] Numpy documentation available at *https://numpy.org/doc/*