Second Pset on Reinforcement Learning

Leonidas Bakopoulos 2018030036

April 22, 2023

Introduction

The purpose of this problem set, was for the student to familiarize with/compare online decision-making strategy algorithms, in an non-stationary environment, using a given "traffic dataset". In order to achieve the previously mentioned goal, the Multiplicative Weights (MW)algorithm was implemented both in an expert and a bandit environment, tested but also compared with the previously elaborated UCB algorithm (see first pset).

Multiplicative Weights (Experts)

First of all, the MW in "Experts" environment was implemented. In this environment is suggested that in every round (although a single server is selected), the algorithm is informed of the load of every server. Using this extra information, the algorithm can calculate the loss (loss(t)= load of every server(t) - load of the best server (t), noting that as "best" server is considered the one with the minimum load). In the next step, the algorithm updates the weight/probability of **every** expert (using the formula $w* = (1-\eta)^{l_i^t}$, where l_i^t is the loss of expert i in round t, and $i \forall \|expert\|$). Then a new expert is stochastically selected, based on the mentioned probabilities, and that decision process/ learning repeats itself for all episodes.

Multiplicative Weights (Bandits)

In the Adversarial Bandits environment, the algorithm is not informed of every server's load for each round t. In order for the algorithm to be adjusted to the "new" environment, some changes were needed. First of all, in round t the update function is $w = w(1-\eta)^{l_i^t}$, with $l_i^t = \begin{cases} \frac{l_i^t}{q_i} & \textbf{for picked arm i} \\ 0 & \forall j \neq i \end{cases}$ (1). This means that for each round t the only weight that is changed is the one that belongs to the chosen server. The second difference, is the calculation of $q_i = (1-\epsilon)p_i^t + \frac{\epsilon}{k}$ in order to add some uniform exploration for every arm. This q_i is used both in weight update (1) and in stochastic selection (instead of p_i). The intuition behind the (1), is that this update should have been in every round (just like in the previous algorithm), but now the loss cannot be calculated. By dividing with q_i, we now calculate a sum of the mean loss (for every round) instead.

Note

For both algorithms, is used $\eta = \gamma = (\frac{log(k)}{T})^{\frac{1}{2}}$[1]. Gamma = $\frac{k}{T}$ is also used (as [2] suggests with $p = T$) with no differences in the result, as we can see by comparing figures 1 and 2. Even though theory in general suggests $\gamma \epsilon [.1, 1]$, in our case a $\gamma > .15$ reduces the performance of the algorithm.

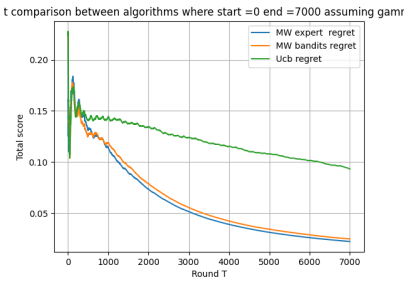

Figure 1: Assuming $gamma = \frac{k}{T}$

Experiment

In order to compare the performance of the previously mentioned algorithms, a **non stationary** dataset, that stores the load of k=30 servers for T = 7000 episodes is used. The algorithms (Multiplicative Weights in Experts,

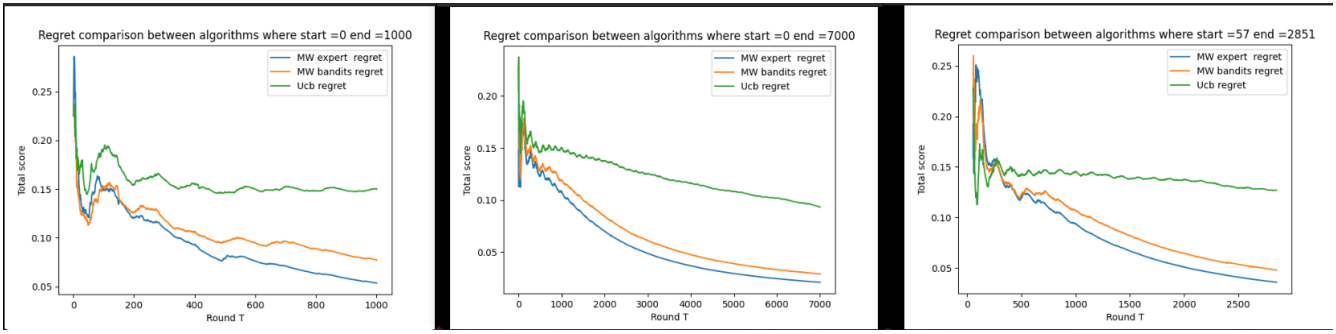

Figure 2: First Experiment

Multiplicative Weights in Bandits and UCB- see appendix-) were asked to find for each epoch the server with the lowest load, based on the previous observation (note again that the data is **not IID**). Two types of experiments were performed. In the first type, the algorithms were implemented as theory suggest and were tested in three scenarios. The first scenario starts at t=0 and has horizon T=1000, the second has the same start but a horizon of T=7000 and the last one has a random start and a random T. In the second experiment, the MW in Bandits environment, is implemented ignoring the q_i and replacing it with p_i in server selection, and **not** in (1)(but without changing the environment, in every round we can calculate the loss only for the chosen server).

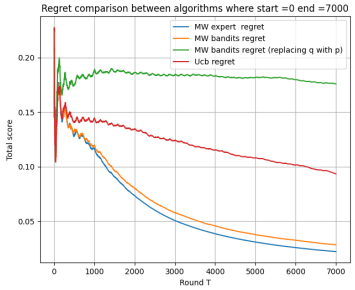

Figure 3: Second Experiment

In the graphs above (Figure 2), the MW applied on the experts environment, yields better results/lower regret than on the bandit environment. Both algorithms perform better than UCB. Theory has proved that the results above have also a mathematical sense. Firstly, the comparison between the MWs (Experts vs Bandits) it is not completely fair, because the algorithms are not implemented in the same environment. In the Experts environment, the algorithm is informed of the loss of every server, whereas in the bandits it isn't. That is a big advantage of the MW experts as it is shown in the figures. Both algorithms, have same order of regret ($T^{\frac{1}{2}}$), something that implies that they have same structure, and differences are caused by the environment. Secondly, as theory suggests,and is also proved by the figures, a deterministic algorithm (ex UCB), is not the ideal choice in a non stationary environment. Last but not least, we can see that UCB, has a sublinear regret (not of the same order as MWs), that verifies that UCB can be trained in this environment, but its performance differs from MWs.
In Figure 3,(results of the second experiment) we can see the performance differences between the various implementations of the "same" algorithm. As it is shown in the graph, the bandits MW (without the q) has the worst behavior. This "almost linear" regret, is mainly caused by the update of the weight, since the new weight of the selected server wasn't divided with neither q_i nor p_i, and is not so much based on the lack of exploration. If the division part was maintained, the algorithm wouldn't have such terrible performance but still the MW experts would have been the best. It is worth mentioning, that the above is an observation that stands only for this particular dataset.

Appendix

UCB's implementation was based on the previous pset implementation. Specifically, in the body of the UCB algorithm, the regret was converted to reward using the expression $reward = -regert$, in order to maintain the idea of maximization of UCB (In an other case, we would have a two value sum for every server - mean loss + exploration - where the first term must be minimized and the second must be maximized). We could also calculate UCB as $ucb_i = load_i - exploration$, and $\pi^* = argmin(ucb_i)$

References

[1]https://courses.cs.washington.edu/courses/cse599i/18wi/resources/lecture5/lecture5.pdf
[2]https://www.eclass.tuc.gr/modules/document/file.php/HMMY284/lecture20.pdf