MVA: Reinforcement Learning (2020/2021) **Homework 4**

Model Selection for Contextual Bandit

Lecturers: *A. Lazaric, M. Pirotta* *(January 9, 2020)*

Solution by AMEKOE Kodjo Mawuena

Instructions

- The deadline is **February 5, 2021. 23h00**
- By doing this homework you agree to the *late day policy, collaboration and misconduct rules* reported on Piazza.
- **Mysterious or unsupported answers will not receive full credit**. A correct answer, unsupported by calculations, explanation, or algebraic work will receive no credit; an incorrect answer supported by substantially correct calculations and explanations might still receive partial credit.
- Answers should be provided in **English**.

1 Model Selection

In this homework, we look into the problem of model selection for bandit. For example,

- Optimizing the exploration rate in an ϵ-greedy algorithm
- Learning the most effective representation of a problem (e.g., UCB v. LinUCB, or selection among multiple linear representations)

These are just two examples of possible applications.
In general, we assume that we are given a set of M bandit algorithms. We aim to design a meta algorithm (or *master algorithm*) which selects at each round which base algorithm to play. The goal is to ensure that the performance of the master is not far away from the best base algorithm had it been run separately. Denote by $R_i(T)$ the regret bound of base algorithm i, the objective is to design a master algorithm having regret bound $\widetilde{O}(poly(M)R_{i^\star}(T))$, where $R_{i^\star}(T) = \min_i R_i(T)$. This problem has received a lot of attention over the years [Agarwal et al., 2012, 2017, Singla et al., 2017], with a surge of interest this year [Abbasi-Yadkori et al., 2020, Pacchiano et al., 2020b,a].

While this idea applies to many scenarios, we consider the problem of selecting a representation for LinUCB. We start reviewing the setting of linear contextual bandit. At each round t, the algorithm receives a context $x_t \in \mathcal{X}$ (we assume the contexts are drawn i.i.d. from $\mathcal{X}$) and needs to select an action $a_t \in \mathcal{A}$ to play (we assume $\mathcal{A}$ is finite). The reward associated to a context-action pair (x, a) is a linear function of a set of features in $\mathbb{R}^d$: $r(x, a) = \langle \phi(x, a), \theta^\star \rangle$. The vector $\theta^\star \in \mathbb{R}^d$ is unknown and the observed reward at time t is a noisy version of the true reward: $r_t(x, a) = r(x, a) + \eta_t$ with η_t being a conditionally zero mean σ-subgaussian random variable. The objective is to control the pseudo-regret: $R(T) = \sum_{t=1}^{T} \langle \phi(x, a^\star) - \phi(x, a_t), \theta^\star \rangle$. LinUCB algorithm suffers a regret at most $\widetilde{O}(\log(\det(\Sigma_t)/delta)\sqrt{t}) \leq \widetilde{O}(d\sqrt{t})$ for any $t \leq T$, where $\Sigma_t = \sum_{i=1}^{t} \phi(x_i, a_i)\phi(x_i, a_i)^\top + \lambda I$ is the $(\lambda > 0)$-regularized design matrix. We now assume that the mapping ϕ is unknown but we have access to a set of candidates $F = (f_i)_{i \in [M]}$. We can instantiate a version of LinUCB for each representation $f \in F$. Denote by $B_i = LinUCB(f_i)$. The new interaction loop is as follows. At each round t, the master observes a context x_t and needs to select the base algorithm B_t to play among the M variants of LinUCB. B_t is executed, i.e., B_t selects the action a_t to play and a reward $r_t(x_t, a_t) = \langle \phi(x, a), \theta^\star \rangle + \eta_t$ is observed. The master and B_t are updated using the observed reward. It is common to assume that at least one mapping is realizable

$$\exists f^\star \in F, \omega \quad : \quad \forall x, a, \; r(s, a) = \langle \phi(x, a), \theta^\star \rangle = \langle f^\star(x, a), \omega \rangle$$

We provided an initial code implementing a linear representation and a bandit problem. The goal is to investigate the approach in the paper [Pacchiano et al., 2020a] and implement their algorithm (Algorithm 1) in this setting.
You can play with the code to test your implementation. We provide a simple entry point in file `main_simple.py` to test your implementation on small problems.
Consider the following setting for the final experiment (already implemented in the file `main_final.py`). We consider a finite set of contexts and actions: $|\mathcal{X}| = 200$, $|\mathcal{A}| = 20$. Let $|F| = 8$ and $f_8 = f^\star$ with $d_8 = d^\star = 20$ (i.e., the realizable representation). The others are such that

- f_1, f_2, f_3 are nested representations of $f^\star$ with size $d_1 = 5$, $d_2 = 10$ and $d_3 = 25$ obtained by selecting the first d_i features when $d_i < d^\star$ and padding random features when $d_i > d^\star$.
- f_4, f_5, f_6, f_7 are randomly generated feature of dimension $3, 7, 16, 30$.

Let $T = 50000$, $\sigma = 0.3$, $\delta = 0.01$ and $\lambda = 1$. Remember to average the regret over multiple runs.
Questions:

1. Start from implementing LinUCB (a.k.a. OFUL). Use the slides or any other material as reference for the implementation (e.g., Section 6.1 in [Pacchiano et al., 2020a]). Suggestion: use Sherman–Morrison formula for a more efficient implementation.
2. Understand the "Regret Bound Balancing and Elimination Algorithm" (RegBalAlg) in [Pacchiano et al., 2020a] (Algorithm 1).

   ! It's a long document, focus only on the parts relevant for the homework. Section 4 should be enough for implementing the algorithm. Clearly, previous sections are necessary to better understand the setting and the notation.

   (a) Could you briefly explain the idea behind the algorithm?
   (b) Could you explain the elimination condition? What does the set $\mathcal{W}$ contain?
3. Implement the algorithm "Regret Bound Balancing and Elimination Algorithm" (RegBalAlg) for the mentioned problem (see provided code).
   (a) Compare the RegBalAlg algorithm versus LinUCB on each representation (i.e., B_i for $i \in [M]$). Report the a picture with the pseudo regret of each algorithm.
   (b) What upper-bound did you use for implementing RegBalAlg?
   (c) Is any representation eliminated? Is the optimal representation the only not eliminated? Try to explain what you observed.
   (d) Plot the regret of RegBalAlg and the one of the optimal representation. If you see a sudden change in the regret of RegBalAlg, could you explain why this happens?

2 Answers

1. See the *algorithms.py* file.
2. (a) The idea behind the regret bound balancing algorithm is to consider M base learner algorithms (representations) and to perform model selection by preserving the regret upper bound. To achieve this goal, the algorithm tries to compete a base learner with an optimal algorithm (unknown), but for which we can compute the lower and upper bound of it's expected reward ($\mu^\star$) up to the round t in stochastic context(over all contexts). So at given round t, base partner that didn't respect (balanced) these bound is eliminated.

(b) A learner i is eliminated at the round t if the upper for $\mu^\star$ from i contradicts the lower bound from any other learner j ie we have:

$$\mu^\star \leq \frac{U_i(t)}{n_i(t)} + c\sqrt{\frac{\log(M \log n_i(t)/\delta)}{n_i(t)}} + \frac{R_i(n_i(t))}{n_i(t)} \tag{1}$$

,

$$\mu^\star \geq \frac{U_j(t)}{n_j(t)} - c\sqrt{\frac{\log(Mlnn_j(t)/\delta)}{n_j(t)}} \quad \text{for any} \quad j \tag{2}$$

and

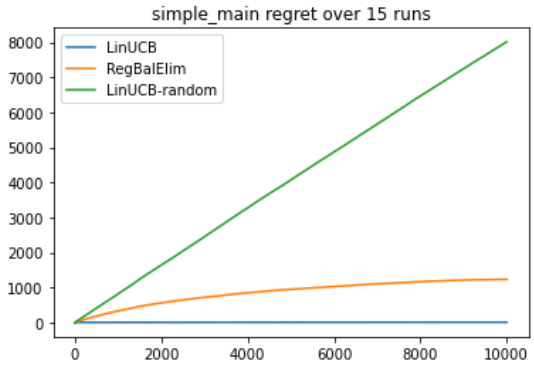
$$\frac{U_i(t)}{n_i(t)} + c\sqrt{\frac{\log(M \log n_i(t)/\delta)}{n_i(t)}} + \frac{R_i(n_i(t))}{n_i(t)} \leq \frac{U_j(t)}{n_j(t)} - +c\sqrt{\frac{\log(M \log n_j(t)/\delta)}{n_j(t)}} \tag{3}$$

where

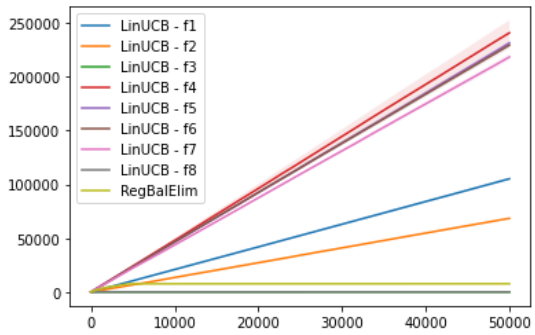
- $U_i(t)$ is the total reward accumulated by the learner i up to the time or the round t
- $n_i(t)$ the number of time the learner i was played after t rounds
- $R_i(n_i(t))$ the upper bound of the learner i

$W = \{i \in [M] : \forall t \in [T] \quad Reg_i(t) \leq R_i(n_i(t))\}$ is the set of well-specified learners.
With $Reg_i(t)$ the pseudo-regret. So W a set containing the learners whose pseudo regret at any time is less than or equal to their regret upper bound.

3. (a) In this part we compare the RegBalAlg algorithm and the LinUCB in on the set of the file *main_simple.py* and *main_final.py*.



(a) Comparison of LinUCB, RegBalAlg and Random-UCB



(b) Comparision of RegBalAlg and LinUCB for each representation

Figure 1: Comparison of the pseudo regret LinUCB versus RegBalAlg

The Figure 1 (a) shows the pseudo (average) regret of 15 runs of RegBalAlg, Linear UCB and a random representation for Linear UCB. The idea of this comparison is to make sure sure that our code runs smoothly.
(b) the average pseudo regret and the standard error over 5 runs of Linear UCB and RegBalAlg on our main example.

(b) For the implementation of the RegBalAlg, we consider the upper bound such that (the symbols used in the course):

$$Reg(T) \leq \sum_{t=1}^{T} \alpha_t \sqrt{\phi(u,a)^\top \Sigma_t^{-1} \phi(u,a)} \tag{4}$$

,

$$\alpha_t = \sqrt{2\sigma^2 \log\left(\frac{det(\Sigma_t)^{1/2} det(\lambda I)^{-1/2}}{\delta}\right)} \tag{5}$$

and we set the absolute constant c to one (1) for all our implementation.

It is important to note that in order to avoid certain calculation problems (such as division by zero), it is assumed that each learner has been played at least once before a possible elimination.

(c) Out of the 5 runs of RegBalAlg, all representations have been eliminated except f_3 and f_8.

For the randomly generated representations f_4, f_5, f_6, f_7 we can see that the pseudo regret decreases with increasing number of feature d. (Figure 1 (b)).

Representations deriving from the optimal f_8 by reducing the number of features f_1, f_2 have also been eliminated by RegBalAlg.

Thus we can say that the farther the representation is from optimal (random) and with a small feature size, the more it can be eliminated by RegBalAlg. However, in the neighborhood of the optimal representation RegBalAlg may have difficulty in selecting a single representation (the case of f_3) as we could see below:

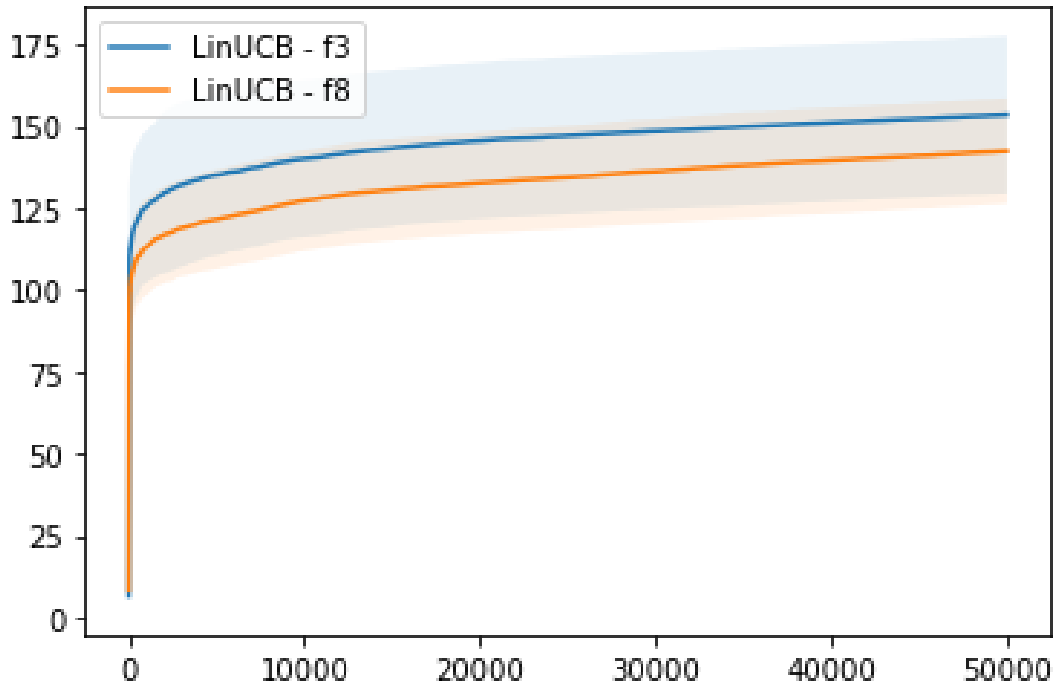


Figure 2: Comparison of the pseudo regret LinUCB for representations f_3 and f_8

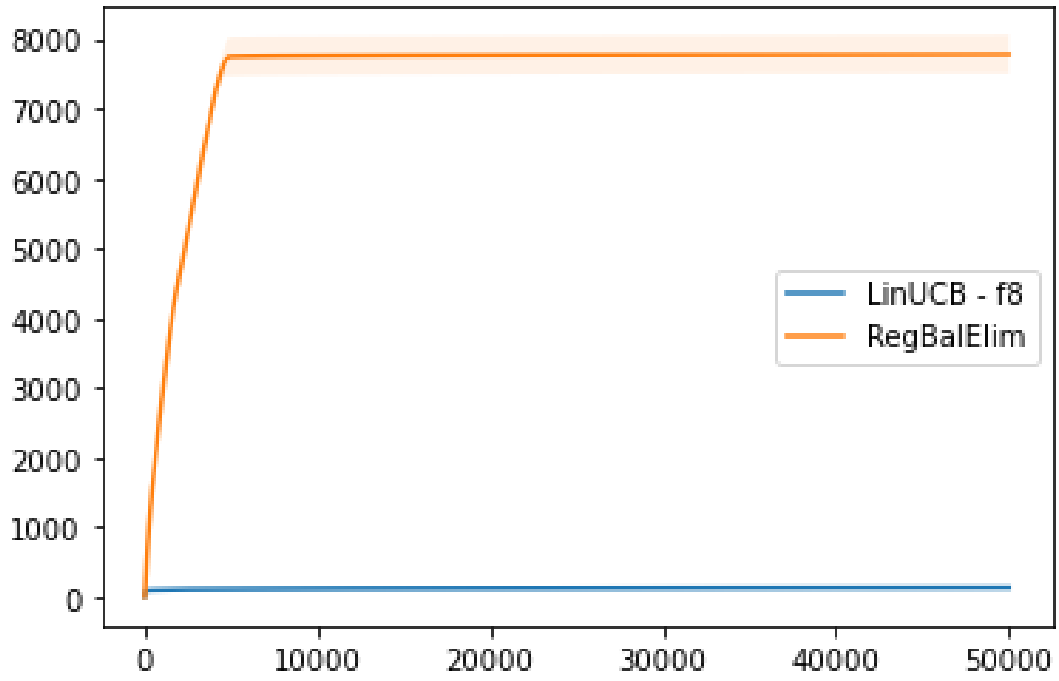


Figure 3: Comparison of the regret for LinUCB (f_8) and RegBalAlg

(d) With the Figure 3 , we can see that there is a big jump between the regret of the RegBalAlg and that of the optimal representation. This was hidden by the scale in Figure 1. The large regret of the RegBalElim algorithm can be explained by the fact that many non-optimal representations (learners) were played over a large number of rounds (t) before being eliminated. RegBalAlg is therefore not a good estimator of optimal regret, but a way to select the right learners.

3 Conclusion

This exercise allowed us to review the linear UCB for a contextual bandit problem. It also allowed us to study the Regret Bound Balancing and Elimination Algorithm (RegBalAlg) which is a model selection algorithm recently proposed by Abbasi-Yadkori et al. [2020] and which does not require any hypothesis on optimal regret.

References

Yasin Abbasi-Yadkori, Aldo Pacchiano, and My Phan. Regret balancing for bandit and RL model selection. *CoRR*, abs/2006.05491, 2020.

Alekh Agarwal, Miroslav Dudík, Satyen Kale, John Langford, and Robert E. Schapire. Contextual bandit learning with predictable rewards. In *AISTATS*, volume 22 of *JMLR Proceedings*, pages 19–26. JMLR.org, 2012.

Alekh Agarwal, Haipeng Luo, Behnam Neyshabur, and Robert E. Schapire. Corralling a band of bandit algorithms. In *COLT*, volume 65 of *Proceedings of Machine Learning Research*, pages 12–38. PMLR, 2017.

Aldo Pacchiano, Christoph Dann, Claudio Gentile, and Peter L. Bartlett. Regret bound balancing and elimination for model selection in bandits and RL. *CoRR*, abs/2012.13045, 2020a.

Aldo Pacchiano, My Phan, Yasin Abbasi-Yadkori, Anup Rao, Julian Zimmert, Tor Lattimore, and Csaba Szepesvári. Model selection in contextual stochastic bandit problems. In *NeurIPS*, 2020b.

Adish Singla, Seyed Hamed Hassani, and Andreas Krause. Learning to use learners' advice. *CoRR*, abs/1702.04825, 2017.