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

Model Selection for Contextual Bandit

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

Solution by Pierre Ebert

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.

   See the algorithms.py file for the implementation and Question 3 for the commentary.

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?

   The RegBalAlg algorithm is based on the concept of *regret bound balancing*. RegBalAlg is used on the problem $(\chi, \mathcal{A}, \prod)$, with χ the context space, $\mathcal{A}$ the action space and $\prod$ the policy space. RegBalAlg is initially given a set of M *learners*, i.e. learning algorithms working on the same problem $(\chi, \mathcal{A}, \prod)$. Each learner has a presumed regret bound R_i which is a function of the number of times this learner was used. RegBalAlg selects a learner at each round with the objective of making all the presumed regret bounds approximately equal. The learner is then played and the reward used to update the internal statistics. The idea is that the actual accrued regret of each learner will not exceed its presumed regret and, as RegBalAlg strives to keep these regret bounds equal, the overall regret of the algorithm cannot be more than M times worse than if the learner with the best presumed regret bound had been selected for all rounds.

   (b) Could you explain the elimination condition? What does the set $\mathcal{W}$ contain?

   RegBalAlg is perhaps more interesting when the real properties of the learners are not well known. For example, the presumed bound of a learner could be badly specified. In this case, an elimination condition is evaluated for each selected learner, in order to identify and eliminate misspecified learners using the misspecification test.

   In this test, the time-average rewards of the selected learner i is compared to that of an other learner j. Let μ be the deterministic but unknown optimal average reward for which a lower bound can be obtained, involving B_j and the time-average reward of learner U_j:

   $$\mu \geq U_j(t) - B_j(t) \tag{1}$$

Furthermore, an upper bound can be obtained using the presumed bound R_i of learner i:

$$\mu \leq U_i(t) + B_i(t) + R_i(n_i(t))/n_i(t) \tag{2}$$

If at any round t, we have the upper bound of μ given by learner i (Equation 2))which contradicts the lower bound given by a learner j (Equation 1),

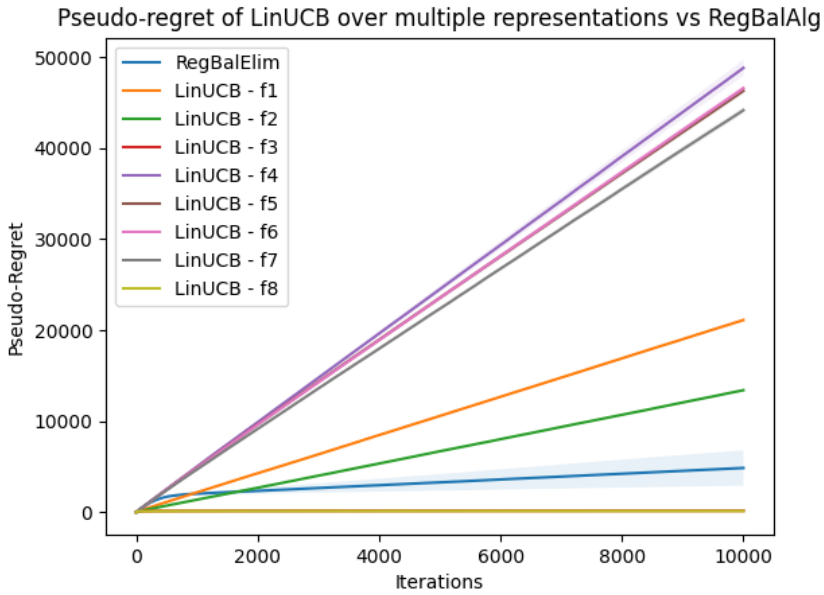
$$U_i(t) + B_i(t) + R_i(n_i(t))/n_i(t) \leq U_j(t) - B_j(t)$$

it can be concluded that the upper bound given by learner i is false and that learner i is badly specified. The learner i is then eliminated from the active learners set.

The learners are then progressively eliminated, keeping only well specified learners. The selection method described in the previous method ensures that the concept of *regret bound balancing* is followed.

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.



The figure above show the pseudo regret of each algorithm (averaged over 30 runs). We have 8 LinUCB algorithms which each have their own representation and one RegBalAlg algorithm, which progressively eliminates misspecified learners, i.e. cease to use the LinUCB algorithms which break their pseudo-regret bounds. We can see that RegBalAlg performs better than all the LinUCB algorithms except for LinUCB with representations 3 and 8 (on the graph, the regret curve of LinUCB - 3 is hidden behind that of LinUCB - 8).

   (b) What upper-bound did you use for implementing RegBalAlg?

I used the following upper regret bound, from Algorithm 1 of Section 4 of Pacchiano et al. [2020a].

$$\frac{U_i(t)}{n_i(t)} + \frac{R_i(n_i(t))}{n_i(t)} + c\sqrt{\frac{\ln(M\ln(n_i(t))/\delta)}{n_i(t)}}$$

with $c = 1.7$ from Lemma A.1.
I replaced $R_i(n_i(t))$ with its bound as shown in Section 6.1 of Pacchiano et al. [2020a]:

$$R_i(n_i(t)) \leq 2\beta_{max}\sqrt{dt\Big(1+\frac{L^2}{\lambda}\Big)+\ln\Big(\frac{d\lambda+tL}{d\lambda}\Big)}$$

with L the bound of the action norm and λ a regularisation parameter. Finally, $\beta_{max} = \max_t \beta_t$, with the following definition of β_t :

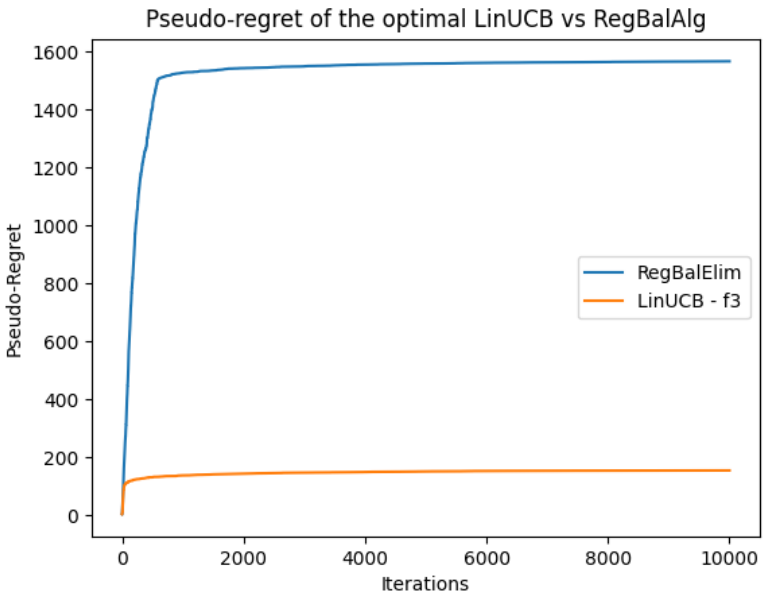
$$\beta_t = \sqrt{\sigma^2 d \ln\Big(\frac{1+tL^2/\lambda}{\delta}\Big)} + \sqrt{\lambda}S$$

with S the bound of the parameter norm and σ the variance parameter of the noise random variable.

(c) Is any representation eliminated? Is the optimal representation the only not eliminated? Try to explain what you observed.

Initially, the following representations are included in the learner active set: [1, 2, 3, 4, 5, 6, 7, 8]. As the RegBalAlg progresses, it eliminates learners which are *misspecified*,i.e. have bad performance and break the presumed regret bound. At the end of 50000 iterations, only two learners were usually retained ([3, 8]). For some runs, one of the two learners was eliminated but generally both learners were retained. From the Figure above, it is clear that learners 3 and 8 have by fare the lowest pseudo-regrets and it therefore makes senses that they are the only ones retained. As their pseudo-regrets are very close (indistinguishable on the figure above), the algorithm struggled to eliminate one of them and select the final optimal learner.

(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?



The Figure above shows the regret of RegBalAlg vs. that of the optimal learner (LinUCB with the third representation). Clearly, it is expected that RegBalAlg's regret is higher than the optimal learner because RegBalAlg has had to eliminate the other learners before settling for the optimal learner. There is a clear discontinuity in the pseudo-regret of RegBalAlg after 1000 iterations. In the present example, RegBalAlg was not able to obtain a single final

learner, retaining 3 and 8, but this is also expected as the performance of both learners are extremely close. The discontinuity in the curve of RegBalAlg is likely to be the point at which RegBalAlg eliminated the last sub-optimal learner and settled for learners 3 and 8, its regret thus reaching the well-specified bound.

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.