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

Model Selection for Contextual Bandit

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

Solution by Mahdi KALLEL

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.

   Please check the LinCUB code. Where we implement the OFUL Algorithm with Sherman Morrison Formula for matrix inversion.

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?

   RegBalAlg is a general purpouse model selection master algorithm. It can be combined with multiple base bandit algorithms to find the best candidate with a regret competitive to that of the best candidate. The algorithm follows an elimination scheme that maintains over time a set $\mathcal{W}$ of active learners
   The way it works is that for each round t that the master algoritmh interacts with the environment it has to choose a learner which in turn will choose an action. We keep track of each learner's separate regret. The algorithm is an action elimination scheme that maintains over time a set of active learners, and undergoes an elimination procedure.
   This procedure relies on a statistical test to detect when a learner is miss specified. We call learner i well-specified if it's regret is below the theoretical regret bound,for all the previous time steps, with high probability.

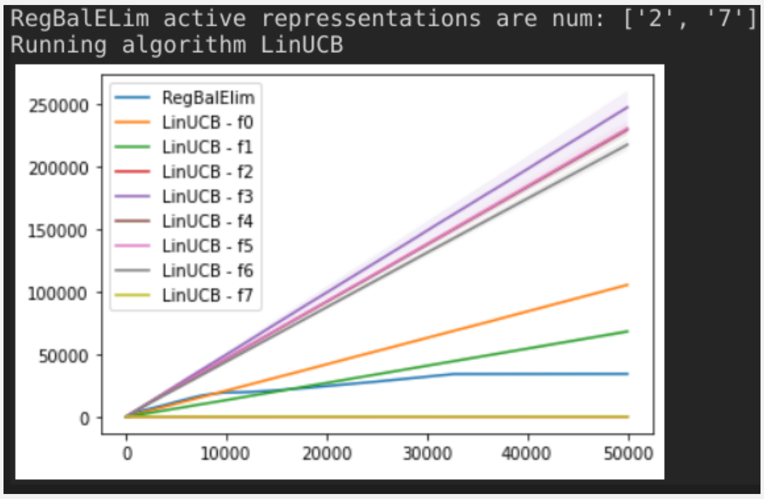
   (b) Could you explain the elimination condition? What does the set $\mathcal{W}$ contain?

$\mathcal{W}$ is a time dependant set $\mathcal{W}_T$ that contains all the "active learners", after the master algorithm has interacted T times with the environment. RegBalAlg uses a misspecification test to identify and eliminate misspecified base learners. This test compares the time average rewards $\frac{U_i(t)}{n_i(t)}(t)$ and $\frac{U_j(t)}{n_j(t)}(t)$ achieved by two base learners i and j. More precisely the bound's on the reward upper bound for learner i $=$ $\frac{U_i(t)}{n_i(t)} + c\sqrt{\frac{ln(Mlnn_i(t)}{n_i(t)}} + \frac{R_i(n_i(t))}{n_i(t)}$
and the reward lower bound for learner j $\frac{U_j(t)}{n_j(t)} - c\sqrt{\frac{ln(Mlnn_j(t)}{n_j(t)}}$
If at any round the upper bound of learner i contradicts the lower bound for learner j then it's eliminated from the set $\mathcal{W}$ the aim of this scheme is to eliminate "miss specified learners" that do not respect their theoritical regret $R_i(n_i(t)$.

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.

In the plot below we plot the performance of RegBalElim trained using 8 different representations for LinUCB, and we compare it to the performance of each of these learners separately. We see that from it's early stage, the algorithm is able to eliminate non promising learners keeping always keeping a regret below f4,f5,f6. The algorithm requires almost 30,000 iterations to finally reach a constant regret.



```
RegBalELim active repressentations are num: ['2', '7']
Running algorithm LinUCB
```

Figure 1: Regret of RegBalElim compared to each separate learner

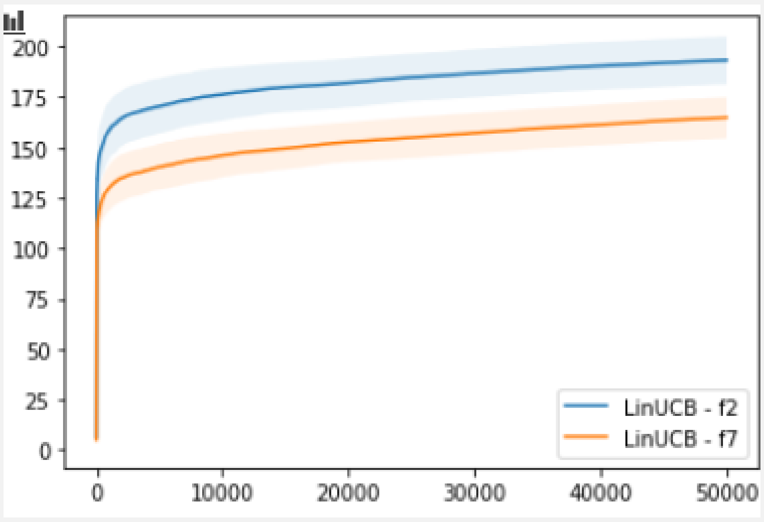


Figure 2: Regret of LinUCB on the final (kept) representations

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

For the OFUL algorithm, the paper recommends using the regret $Reg(T) \leq 2\beta_{\mathbf{max}}\sqrt{dT(1+\frac{L}{\lambda})ln\frac{d+TL}{d}}$ we find that this bound was too loose and didn't work. For our implementation we used the more constrained bound : $Reg(T) \leq 2\sum_1^T \beta_t ||a_t||_{\Sigma_t^{-1}}$. It's worth noting that we tried varying the confidence parameter "c" from 1 to 0.01 and we didn't notice a major effect on the final regret.

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

From figure 1. We see that only the representation 2 and 7 are kept after 50,000 rounds. f0,f1 dimensions are not sufficient to solve the problem. f3..f6 are just noisy representations and it's obvious they will have a much higher regret and thus are eliminated first. Representation 7 being the real optimal one. Representation 2 contains all of f7 dimensions + 5 dimensions of noise. LinUCB is able to detect the noisy dimensions and assign them a zero weight. Thus f2 and f7 are sufficient to solve the bandit problem.

(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?

Each sudden change in the regret corresponds to dropping one of the "missspecified" learners. The more the learner is miss specified the more regret we encure because of it and thus the higher it contributed to the regret "slope", by dropping it is like taking away this learners slope from the RegBalElim regret.

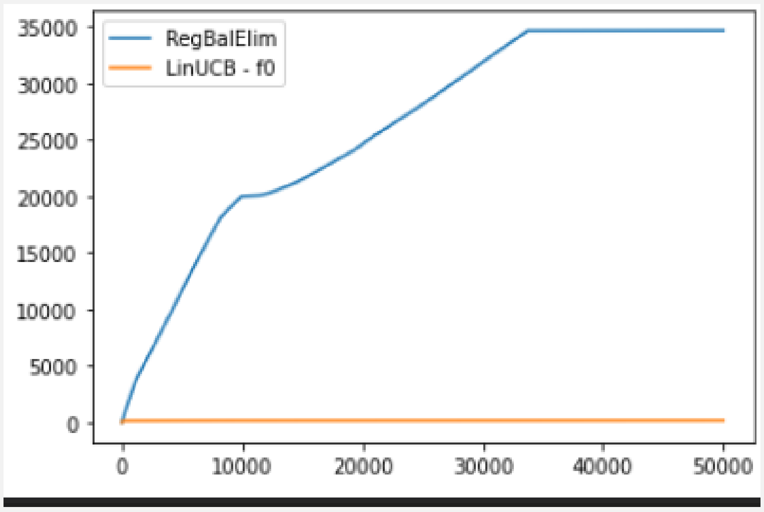


Figure 3: Regret of RegBalElim to the good representation

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.