Stochastic Linear Bandits An Empirical Study

Students: Ognjen PERIC, Antoine PEYRONNET,
Lecturer: Claire Vernade, Contact me on Slack if anything looks weird, or find my email on my website

Abstract:

We compare different linear bandit method in RL applied to a Multi Armed Bandit Problem. Playing on different factors such as the number of those Arm, and the proper parameters of each strategies makes us understand empirically why some strategies (Linear Epsilon Greedy, Lin UCB, Lin Thompson Sampling) are better to adopt and why depending on the context

Problem I: Linear Epsilon Greedy Algorithm

1.1: Experiment

Both experiments (fixed or randomly chosen ,e.g **iid**) share a common can be set up as follows in 4 stages:‘

1. In both cases, the arms are initialized with random characteristics (examples of vectors in a d-dimensional space). For the randomnly chosen case, we do this at each step.
2. Regrets are generated linearly based on the characteristics of the arms, but with some random noise to mimic real-world environments. Regrets can be determined by a linear model with unknown coefficients (e.g., $\theta^T x$ where x is a feature vector of an arm and θ is a weight vector?).
3. The algorithm we use selects an arm according to the algorithm's policy
4. we add a stopping criterion he experiment may end after a predefined number of iterations (e.g., 1000 steps).

We tested espilon-greedy algorithm accross dimensions $\in \{3, 10, 20, 30\}$, exploration rate $\in \{0, 0.1, 0.2, 0.5\}$, but we fixed the horizon $T = 1000$, and averaging over $N = 10$ Monte Carlo simulations. The regrets are sampled with noise $\sigma^2 = 1$. At last the uniform sampling is included as a baseline.

Metrics and Results

The performance of the algorithm can be measured by cumulative regret. The sum of regrets obtained by the algorithm over time. This gives an idea of the algorithm's effectiveness in exploring and exploiting the best actions.

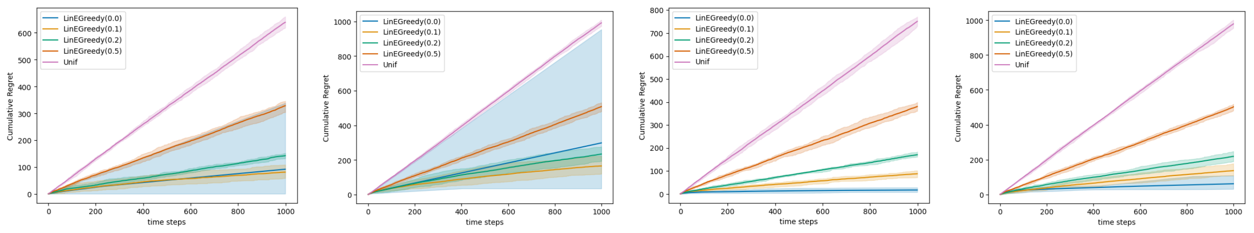
Algorithms and Performances

- LinUniform: This algorithm selects actions randomly from the available arms, without using any learned information, and does not update any model based on received rewards.
- LinEpsilonGreedy: This algorithm chooses actions using an epsilon-greedy strategy where the agent explores randomly with probability *epsilon* and exploits the arm with the highest expected regret based on least squares estimates of the regret model. We say

- LInUCB This algorithm uses an Upper Confidence Bound (UCB) approach where actions are chosen based on the current estimate of the rewards, with an exploration bonus computed using the confidence bounds on the regret estimates, adjusted over time using least squares. (see section 2)
- LinTS (Thompson Sampling): This algorithm samples from the posterior distribution of the regret parameters (assumed to be normally distributed) and chooses the action that maximizes the expected regret based on the sample, updating the regret model using least squares and posterior distributions.(see section 2)

Experience and Results

Impact of dimension As a remark, we just observed that the confidence intervals of the cumulative regrets were diverging as the the dimension was getting higher. We guess without a mathematical proving to support this idea that this is probably due to the fact that since the dimension is higher it's more complex to estimate θ and **for the rest of the results we will stay at dimension $d = 3$.**



(a) Features fixed ($K = 7$) **(b)** Features fixed($K = 100$) **(c)** Features set randomly ($K = 7$) **(d)** Features set randomly ($K = 100$)

Figure 1: Comparison of results for different scenarios based on feature settings and arm counts.

We vary the number of arms K from 7 to 100. Let's regroup the results of the extremes $K = 7$ and $K = 100$ in the following array as we explain here: For each experience: changing K, fixed or iid, we collected by hand 50 results of the experience **fixed the actions at the beginning** and we use **7 arms** (resp **100 arms**) and note the strategy that had the least regret (so is the most performant)

Number of arms	Fixed actions	Not Fixed actions
7	[pure greedy] and [lin Greedy with $\varepsilon = 0.1$]	[pure greedy]
100	[lin Greedy with $\varepsilon = 0.1$]	[pure greedy]

In the case $K = 7$ arms, it came out that the first strategy used was split between $\varepsilon = 0.1$ greedy strategy and the pure greedy strategy. $\varepsilon = 0.1$ greedy strategy is as performant as the pure greedy strategy.
Pure greedy strategy becomes less and less performant as we increase K (we saw experimentally the results on K = 20, 50, 70, 100 then). See **Fig (b)** for $K = 100$, where the $\varepsilon = 0.1$ greedy strategy stays the best strategy. The pure greedy strategy has a confidence interval that grows bigger and bigger over time which makes it less reliable. That's explainable because as we restrict the number of arms, we increase the probability of the strategy choosing initially the best action initially. As it is pure greedy it will only pick the initial arm and then be near optimal or optimal.

Complexity and Optimization of Matrix Inversion/Update in Lin Epsilon Greedy: We update the covariance matrix this way $B_{t+1} = B_t + uv^T$ and B_{t+1}, B_t stay invertible When we are **not fixing the actions** it turns out that **pure greedy works better than the rest in all cases** and that's explainable since the actions are randomly chosen each step, the pure greedy algorithm then performs as if it was exploring over time. Updating the covariance matrix B_t and its inverse B_t^{-1} requires high complexity: $O(d^3)$. To address this issue, we use the Sherman-Morrison formula:

$$\boxed{\mathrm{B}_{t+1}^{-1} = {B_t}^{-1} - \frac{{B_t}^{-1} u v^\top B_t^{-1}}{1 + v^\top {B_t}^{-1} u}}$$

This update has complexity $O(d^2)$, making the algorithm much more adaptable to higher-dimensional action spaces. (see plotted tendency **Fig 2**)

Runtime comparison for update/inversion methods.

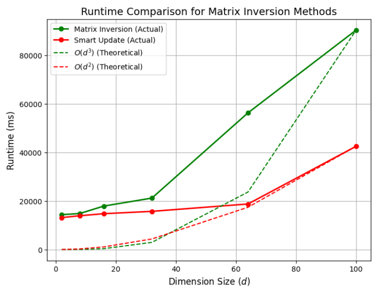
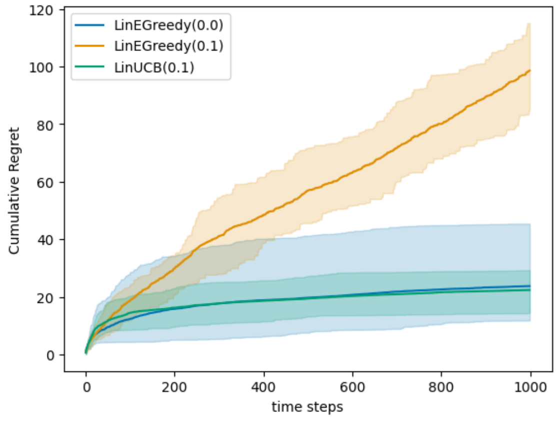


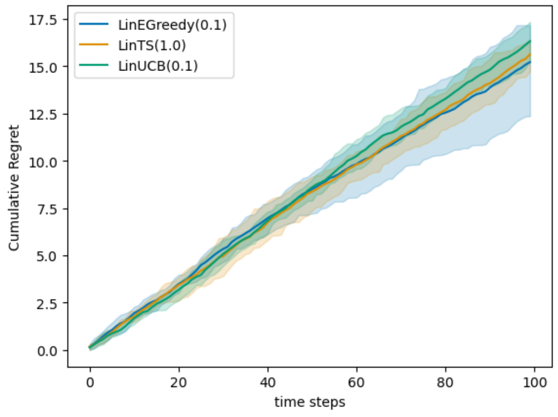
Figure 2

Problem II: LinUCB and LinTS

We are still at $K = 100$. For Thomson Sampling algorithm given a Gaussian prior $\theta \sim \mathcal{N}(0, \kappa^2 I)$ and a reward $r_t = \theta^\top a_t + \epsilon_t$, where $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$, the posterior distribution $\theta \mid a_{1:t}, r_{1:t}$ (where $a_{1:t}$ and $r_{1:t}$ are the sequences of actions and rewards until time t) at the beginning of round t+1 is also Gaussian. Using the Bayesian update for linear regression: $A_t = \lambda I + \sum_{s=1}^{t} a_s a_s^\top$ with $\lambda = \frac{\sigma^2}{\kappa^2}$ $b_t = \sum_{s=1}^{t} r_s a_s$ (Response Vector) $\mu_t = A_t^{-1} b_t$ (Posterior Mean), $\Sigma_t = \sigma^2 A_t^{-1}$ (Posterior Covariance). Thus, the posterior distribution is$\theta \mid a_{1:t}, r_{1:t} \sim \mathcal{N}(\mu_t, \Sigma_t)$ The cumulative regret analysis shows



(a) Performance with IID actions.



(b) Performance with fixed actions.

Figure 3: Comparison of strategies for IID and fixed actions.

that globally **UCB** perform as well as pure greedy algorithm in scenarios with fixed actions, as it effectively adapts to stable environments by sampling from the posterior distribution of the parameter vector $\hat{\theta}$.

In contrast, **LinTS** tends to perform slightly better than the other strategies when actions are generated dynamically but it doesn't make it a defining winner as the confidence bounds of all three strategies in **Fig (b)**