Stochastic Linear Bandits An Empirical Study

Students: Mathias Grau, Alexandre Ver Hulst
Lecturer: Claire Vernade, website

Linear Epsilon-Greedy

Completing the Implementation

In the Linear Bandit environment, the reward r_t at each time step t is modeled as: $r_t = x_t^T\theta + \epsilon_t$, where: $x_t \in \mathbb{R}^d$ is the context vector at time step t, $\theta \in \mathbb{R}^d$ is the unknown parameter vector that we aim to estimate, ϵ_t is noise, typically assumed to be Gaussian noise with mean 0 and variance σ^2.

The objective is to select an action a_t (corresponding to a context vector x_t) to maximize the expected reward. To generate actions, we implemented an action generator that uniformly samples k context vectors from the unit sphere, ensuring diversity with a tolerance rate of similarity. For example, 10 vectors in $\mathbb{R}^3$ are shown in the following figure:

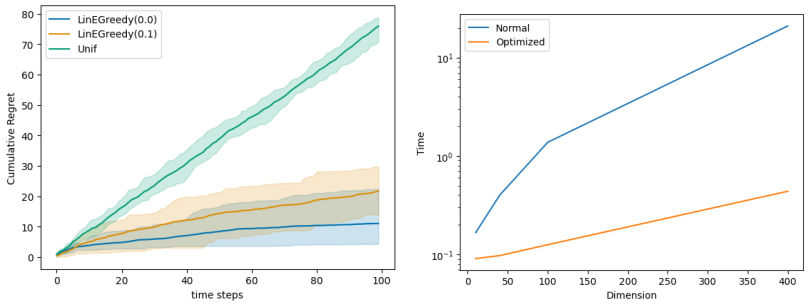
The context vectors $x_1, x_2, \ldots, x_k$ are then compared for their corresponding rewards, and the regret is computed by comparing the selected actions to the optimal ones, based on the largest expected reward.

Implementing Linear Epsilon-Greedy and Testing

For testing, we chose a simple $d = 3$-dimensional problem with $K = 7$ arms and a true parameter vector $\theta = [0.53, 0.59, 0.59]$. We experimented with different values of ϵ (e.g., $\epsilon = 0.0, 0.1$) to compare the algorithm's performance in terms of cumulative reward.

We define regret $R(T)$ as: $R(T) = T\mu^* - \mathbb{E}\left[\sum_{t=1}^{T} \mu_{A_t}\right]$, where μ^* is the reward of the best arm, and μ_{A_t} is the reward of the selected action at time t.

The results, shown in Figure 1a, illustrate the performance of Linear Epsilon-Greedy versus a baseline of Linear Uniform (uniform random action selection).



(a) Performance of the Linear Epsilon-Greedy vs Linear Uniform

(b) Performance optimization of the Linear Epsilon-Greedy for different values of dimension

We observe that the regret for Linear Epsilon-Greedy increases logarithmically with time, as expected from the theoretical result: $R(T) = \mathcal{O}\left(\frac{K\log(T)}{d^2}\right)$ Auer et al. (2002). This confirms that the algorithm's regret grows slower compared to the uniform random selection baseline. However the Epsilon Greedy method seems to be good baseline, we need to implement UCB and Thompson for comparison.

Complexity of the Matrix Inversion Step and Incremental Update

In the Linear Epsilon-Greedy implementation, the matrix inversion step is performed using the pseudoinverse function (`numpy.linalg.pinv`). The computational complexity of this operation is $\mathcal{O}(d^3)$, where d is the dimensionality of the context vector. As d increases, the time required for matrix inversion grows rapidly, making the algorithm computationally expensive for high-dimensional problems. Instead of recalculating the inverse covariance matrix from scratch, we can use the Sherman-Morrison formula to perform an incremental rank-one update. For the covariance matrix update: $\text{cov}_{t+1} = \text{cov}_t + uu^T$, where u is the chosen arm, the inverse covariance matrix can be updated incrementally as: $\text{invcov}_{t+1} = \text{invcov}_t - \frac{\text{invcov}_t\, u\, u^T\, \text{invcov}_t}{1+u^T \text{invcov}_t u}$.

This avoids recalculating the inverse matrix entirely, reducing the complexity from $\mathcal{O}(d^3)$ to $\mathcal{O}(d^2)$, which is significantly more efficient.

To assess to what extent the optimization using the incremental update speeds up the model, we ran experiments for $d = 10, 40, 100, 400$. The results are shown in the figure 1b.

LinUCB and LinTS

Implementation of LinUCB and LinTS

Firstly, we have implemented LinUCB, which implied adding δ as a parameter, δ representing the level of confidence that θ is in the chosen interval. Comparing to LinEpsilonGreedy, the modifications are in the get_action function. Indeed, we implemented β , which is an exploration bonus to find the happy medium between exploration and exploitation. Concretely, β quantifies the uncertainty that we have on the actual θ parameters. To represent that, we have $\beta = \sigma\sqrt{2\log\left(\frac{1}{\delta}\right) + d\log\left(1+\frac{t}{\lambda d}\right)} + \sqrt{\lambda}\|\hat{\theta}\|$, where σ is our variance parameter, d is the dimension of the context vector, t is the timestep, and λ is our regularization parameter. Then for each arm, we compute the sum of the empirical mean (exploitation term) and the exploration bonus (exploration bonus), which is equivalent to $x^T\hat{\theta}_t^\lambda + \|x\|_{(B_t^\lambda)^{-1}}\beta(t,\delta)$, where $B_t^\lambda = \lambda I_d + \sum_{s=1}^t x_s x_s^T$ (x_s being the previous chosen arms). B_t^λ represents the confidence bounds for such linear problems. We then select the arm that has the best score, to optimize our lower bound algorithm, and finding the right compromise between exploration and exploitation.

For the Thompson sampling, the difference resides in the way we choose the arm. Whereas LinUCB had a deterministic way of determining the upper-bounds, and selecting an arm based on a deterministics calculus, based on confidence intervals, LinTS generates at each round a new θ guess based on the last guesses. Then we compute the expected reward for each arm with this θ, and choose the highest score. In this case, the exploration is more diversified, as it depends on a realisation of a random variable.

For Thompson Sampling, what is the posterior at time t?

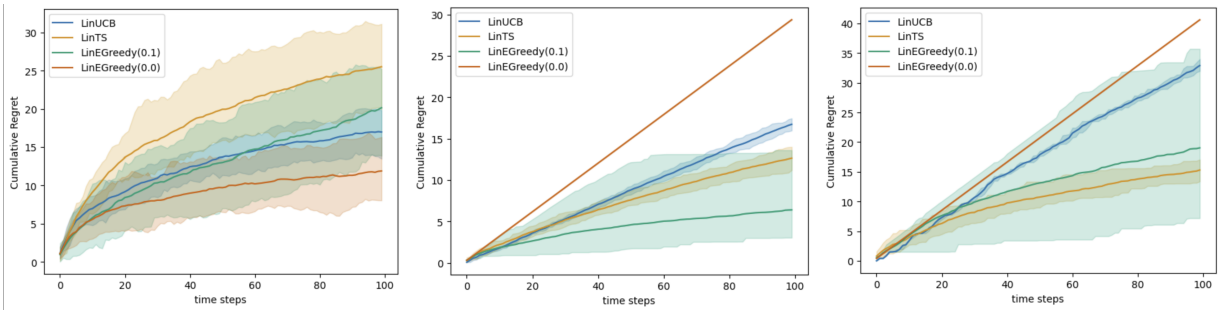
For the posterior distribution, if we take a prior $\theta_* \sim \mathcal{N}(0, \kappa^2\mathbf{I}_d)$, our posterior θ_* follows the distribution $\mathcal{N}(\hat{\theta}_t^\lambda, \sigma^2(B_t^\lambda)^{-1})$, where $\hat{\theta}_t^\lambda$ is our estimation for θ, and as said before, the term $\sigma^2(B_t^\lambda)^{-1}$ represents our confidence level around the $\hat{\theta}_t^\lambda$, so that we can still explore around our estimated value.

Is there a better algorithm ?

We have implemented a music environment, where our parameters are the number of different ads that we can choose (number of actions) *num_ads*, the number of features represents the dimension

of the feature space *num_features*, the standard deviation of noise in rewards *noise_std*, and whereas the possible actions are fixed or iid (the problem type *pb_type*.

Then, we have chosen to run this environment on our different models with the parameters shown in Figure 2a , 2b and 2c. Each model having its own specificities, we expected the following results: when the arms change dynamically, we expected a probabilistic algorithm such as LinTS to perform better, which is what we observe on 2c. Also, the more uncertainty there is, for example with more noise, and more different arms, the more we expect robust algorithms such as LinTS and LinUCB to perform better, whereas LinEGreedy(0.0) is expected to underperform. This is what we observe when we put a noise of 0.5 as shown in 2b, we see that the regret for the LinEGreedy(0.0) is linear, while the others perform well, the best being LinEGreedy(0.1). Finally, when there is less uncertainty, LinEGreedy(0.0) converges faster than the other algorithms, having thus a smaller regret, as shown in 2a. Finally, we conclude from this small study that there is no "best algorithm"



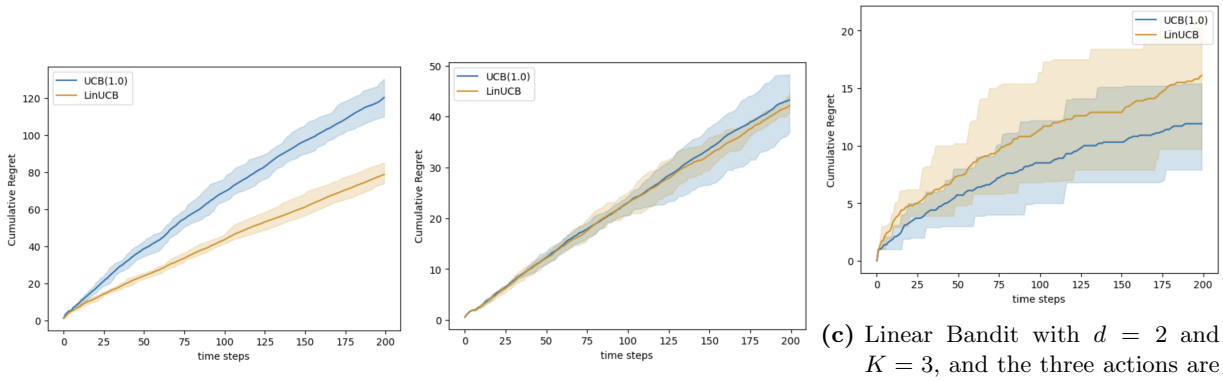
(a) Music Environment with 10 ads, 5 features, 0.1 of noise, and the arms are fixed

(b) Music Environment with 20 ads, 5 features, 0.5 of noise, and the arms fixed

(c) Music Environment with 15 ads, 5 features, 0.3 of noise, and the arms are iid

for the bandit problems ; the best solution depends on the parameters of the problem, and the expected goal.

The role of the action set: Comparison between LinUCB and UCB



(a) Linear Bandit with $d = 3$ and $K = 7$

(b) Linear Bandit with $d = 7$ and $K = 7$

(c) Linear Bandit with $d = 2$ and $K = 3$, and the three actions are fixed and equal to $[1, 0]$, $[0, 1]$, and $[1 - 10^{-4}, 8 \times 0.1 \times 10^{-4}]$

In the first case in Figure 3a, as LinUCB shares information across actions, it is better at such problem. This idea is reinforced in Figure 3b, as we see that LinUCB and UCB are equivalent when K and d are equal. Finally, Lattimore et al. have proven that in pathological cases, such as the actions chosen for Figure 3c, LinUCB converges slower than UCB, and that LinUCB has in this case linear regret.

References

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). *Finite-time Analysis of the Multiarmed Bandit Problem*, volume 47.