DATA 37200: Learning, Decisions, and Limits (Winter 2026)

Lecture 11: Understanding LinUCB

Instructor: Frederic Koehler

Problem Setup: Contextual Linear Bandits

The Setting:

- At time t, we observe a context x_t and a set of arms $[K]$.
- We map each arm-context pair to a feature vector:

$$\phi(x_t, i) \in \mathbb{R}^d$$

- (E.g., $\phi(x_t, i)$ is a vector embedding you get from Gemini API.)
- **Assumption:** The expected reward is linear in these features with an unknown parameter vector $\theta^* \in \mathbb{R}^d$:

$$\mathbb{E}[r_t \mid x_t, a_t = i] = \phi(x_t, i)^\top \theta^*$$

Today's focus: "deriving" LinUCB

- Last class, we covered the inverse gap weighting/SquareCB method for contextual bandits.
- We also introduced LinUCB as an alternative approach for linear contextual bandits.
- For SquareCB, it is helpful to see the proof using AM-GM inequality to understand why the algorithm uses the "inverse gap" weights $p_t(i) \propto 1/(\lambda + \gamma\Delta_i)$.
- The particular form of LinUCB, on the other hand, is more naturally motivated by Bayesian considerations[1].
- Let's see why Bayes leads us to LinUCB.

[1] As explained in [Li-Chu-Langford-Schapire '10] which popularized it.

Postulated noise model

Just like for online ridge, we will "derive" LinUCB algorithm by making guesses about the distribution of the noise which do not end up having to be true in the final theory and applications.

Noise Model: We assume the observed reward r_t is generated by adding Gaussian noise of fixed variance:

$$r_t = \phi(x_t, a_t)^\top \theta^* + \eta_t, \quad \eta_t \sim \mathcal{N}(0, 1)$$

Step 1: The Bayesian Prior

We place a Gaussian prior on the unknown parameters θ^*.

$$\theta^* \sim \mathcal{N}(\mathbf{0}, aI)$$

- **Mean: 0** (we assume no prior directional bias).
- **Variance:** a (represents our prior uncertainty about the parameters).

Defining the Ridge Parameter λ: In Ridge Regression, λ controls the regularization strength. In the Bayesian view, this arises naturally as the ratio of noise variance to prior variance:

$$\lambda := \frac{\sigma^2}{a}$$

Step 2: Deriving the Posterior

Given data up to time $t-1$, the posterior distribution $p(\theta \mid \mathcal{H}_{t-1})$ is Gaussian $\mathcal{N}(\hat{\theta}_{t-1}, \Sigma_{t-1})$.

Precision Update (Inverse Covariance):

$$\Sigma_{t-1}^{-1} = \underbrace{\frac{1}{a} I}_{\text{Prior Precision}} + \underbrace{\frac{1}{\sigma^2} \sum_{\tau=1}^{t-1} \phi_\tau \phi_\tau^\top}_{\text{Data Precision}}$$

Factor out $1/\sigma^2$ to recover the standard "Design Matrix" A_{t-1}:

$$\Sigma_{t-1}^{-1} = \frac{1}{\sigma^2} \left(\underbrace{\frac{\sigma^2}{a}}_{\lambda} I + \sum_{\tau=1}^{t-1} \phi_\tau \phi_\tau^\top \right) = \frac{1}{\sigma^2} A_{t-1}$$

Implication: The posterior covariance is scaled by the noise: $\Sigma_{t-1} = \sigma^2 A_{t-1}^{-1}$.

Step 3: Predictive Distribution for Arm i

What is our belief about the expected reward $f_i = \phi(x_t, i)^\top \theta^*$?
Since $\theta \sim \mathcal{N}(\hat{\theta}_{t-1}, \sigma^2 A_{t-1}^{-1})$, the prediction is also Gaussian:

$$p(f_i \mid \mathcal{H}_{t-1}) = \mathcal{N}(\mu_{t,i}, \nu_{t,i}^2)$$

- **Mean (Estimate):**

$$\mu_{t,i} = \phi(x_t, i)^\top \hat{\theta}_{t-1}$$

- **Variance (Uncertainty):**

$$\nu_{t,i}^2 = \phi(x_t, i)^\top \Sigma_{t-1} \phi(x_t, i) = \sigma^2 \phi(x_t, i)^\top A_{t-1}^{-1} \phi(x_t, i)$$

Note: uncertainty in parameter is inversely proportional to covariance of data (how much of that direction we have seen).

Step 4: Defining α via Confidence Intervals

We construct a $1 - \delta$ confidence upper bound for the mean reward. Let $z_{1-\delta}$ be such that $\Pr_{Z \sim N(0,1)}[Z \leq z_{1-\delta}] = 1 - \delta$.

$$\mathrm{UCB}_i = \mu_{t,i} + z_{1-\delta} \cdot \sqrt{\nu_{t,i}^2}$$

$$\mathrm{UCB}_i = \mu_{t,i} + z_{1-\delta} \cdot \sigma \sqrt{\phi(x_t, i)^\top A_{t-1}^{-1} \phi(x_t, i)}$$

The LinUCB Exploration Parameter α: We can now see exactly what α represents in the algorithm:

$$\alpha = z_{1-\delta} \cdot \sigma$$

Interpretation: If the environment is noisier (higher σ) or we want a lower probability of our UCB failing (smaller δ), we must explore more (higher α) to be confident in our estimates.

Summary: LinUCB Algorithm

Input: λ (regularization), α (exploration).
Initialize: $A_0 = \lambda I, \quad b_0 = \mathbf{0}$.
Loop $t = 1, \ldots, T$**:**

1. Observe context x_t, create features $\{\phi(x_t, i)\}_{i \in [K]}$.
2. Estimate: $\hat{\theta}_{t-1} = A_{t-1}^{-1} b_{t-1}$.
3. **Select Arm (UCB):**

$$a_t = \operatorname*{argmax}_i \left(\phi(x_t, i)^\top \hat{\theta}_{t-1} + \alpha \sqrt{\phi(x_t, i)^\top A_{t-1}^{-1} \phi(x_t, i)} \right)$$

4. Play arm a_t, observe reward r_t.
5. **Update:**

$$A_t \leftarrow A_{t-1} + \phi(x_t, a_t)\phi(x_t, a_t)^\top$$

$$b_t \leftarrow b_{t-1} + r_t \phi(x_t, a_t)$$

Why LinUCB works in an idealized setting

- Suppose for simplicity that the Gaussian noise model and prior describe the true generative model.
- (These assumptions are not needed: see analysis from textbooks or last year.)
- Let $\delta = 1/KT$, then the expected number of rounds the UCB with $\alpha = \sigma z_{1-\delta}$ fails to upper bound the true mean is $O(1)$.
- As in UCB3, we can bound regret by the sum of widths of confidence intervals (next slides).

Analysis: Regret Decomposition

Let i^* be the optimal arm and i_t be the LinUCB choice.

$$
\begin{aligned}
r_t(\text{regret}) &= \langle \phi_t(i^*), \theta^* \rangle - \langle \phi_t(i_t), \theta^* \rangle \\
&\leq \mathrm{UCB}_t(i^*) - \langle \phi_t(i_t), \theta^* \rangle \quad \text{(Optimism)} \\
&\leq \mathrm{UCB}_t(i_t) - \langle \phi_t(i_t), \theta^* \rangle \quad \text{(Greedy Choice)} \\
&= \langle \hat{\theta}_t, \phi_t(i_t) \rangle + \alpha \|\phi_t(i_t)\|_{A_t^{-1}} - \langle \phi_t(i_t), \theta^* \rangle \\
&= \langle \hat{\theta}_t - \theta^*, \phi_t(i_t) \rangle + \alpha \|\phi_t(i_t)\|_{A_t^{-1}} \\
&\leq 2\alpha \|\phi_t(i_t)\|_{A_t^{-1}} \quad \text{(Cauchy-Schwarz)}
\end{aligned}
$$

Analysis: Elliptical Potential Lemma

- We have bounded instantaneous regret by the "width":

$$\mathrm{Reg}_{CB}(T) \leq \sum_{t=1}^{T} 2\alpha \|\phi_t(i_t)\|_{A_t^{-1}}$$

- **Elliptical Potential Lemma:** (use potential function $\log\det A_t$, same trick from ridge analysis)

$$\sum_{t=1}^{T} \|\phi_t(i_t)\|_{A_t^{-1}}^2 \leq 2d \log\left(1 + \frac{T}{d\lambda}\right)$$

- Using Cauchy-Schwarz on the sum:

$$\sum_{t=1}^{T} \|\phi\|_{A_t^{-1}} \leq \sqrt{T \sum \|\phi\|_{A_t^{-1}}^2} \approx \sqrt{T \cdot d \log T}$$

- **Result:** Regret $\approx \alpha\sqrt{Td} \approx \sqrt{Td\log(KT)}$.

Further references

- For the formal regret bound of LinUCB under general rewards, see the Foster-Rakhlin notes, Lattimore-Szepasvári textbook, or course notes from last year.
  - α must be chosen more conservatively which leads to the $d\sqrt{\log(T)}$ bound (compared with what we did in our idealized setting).
- See Ch 21 and Ch 22 of LS textbook for the general, guaranteed to be close to minimax optimal strategy, using Kiefer-Wolfowitz designs. More complicated variants of LinUCB also achieve this.
- Impossible to ensure confidence intervals shrink for more general function classes. (Why UCB is not so easy to generalize.) Related to impossibility results in conformal prediction. May be a HW problem.

Beginning RL

- Recall that a Markov chain is given by a set of states $\mathcal{S}$, and a transition kernel $P : \mathcal{S} \times \mathcal{S} \to [0, 1]$.
- A Markov decision process is the RL version of a Markov chain, where transitions depend on the agent's decisions (choice of action $a \in \mathcal{A}$). Furthermore, the agent receives rewards and its goal is to maximize the sum of (possibly discounted) rewards over all time.
  - Like Markov chain + multiarmed bandit together.
- The mapping from states to actions which the agent uses is called its policy.
- Offline RL: learn a good policy by watching other agents (e.g. humans). Online RL: learn a good policy by directly interacting with the environment.

Some examples of MDPs

- A combination lock.
  - You enter a password in several steps using a dial or keypad.
  - STATE corresponds to your input so far. (e.g. 55-23 as your first two inputs)
  - ACTIONS: reset, input a number (e.g. 1-60), try to open the lock.
  - Receive a reward only if we input correct number (possibly after resetting), input correct password, and then try to unlock.
- Control of a simple rocket.
  - STATE: position, velocity, angular position, angular velocity, mass/amount of fuel left, fin/canard position, ...
  - ACTIONS: steer (move fin/canard), adjust throttle (amount of thrust).
  - Receive a reward if we successfully reach the target destination.
  - Overlap between RL and control theory.

Why RL can be hard?

- State is only partially observed. E.g. solitaire. ("Partially observed MDP")
- Number of states is often very large.
  - E.g. if we are trying to learn to play a videogame, the current screen output may be part of the state.
  - Another example: solving a Rubik's cube. Fully observed, deterministic transitions, but huge number of states.
- Limited/sparse feedback: maybe you receive a prize if you solve a puzzle/problem, but until then you obtain no rewards. Motivates "reward shaping".
- Distribution shift: e.g., train in a simulator, but needs to work in real life.
- ...