

Bandit Learning

Lecturer: Ben Van Roy

Scribe: Nima Hamidi, Amir Abbas Sadeghian

1 Recap

Recall from the first session that we are working on a model for online decision algorithm. This model is an extension of supervised learning. The supervised learning algorithm learns from the inputs x_t and y_t to predict y_t given x_t , but the difference in this model is that this algorithm generates the next data x_t and it observes the corresponding y_t and it gets reward r_t and the objective is to maximize the sum of rewards. An overview of the setting is shown in Figure 1. In the following subsections, we present two important examples of this problem.

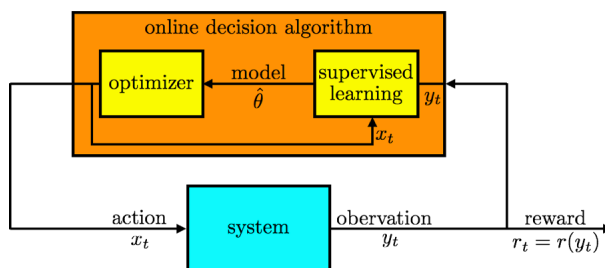


Figure 1: The overview of the bandit learning setting.

1.1 Bernoulli Bandit

- *Action*: $x_t \in \{1, 2, \dots, K\}$,
- *Mean Reward*: $\theta_1, \theta_2, \dots, \theta_K$ corresponding to the above actions,
- *Observation*:

$$y_t = \begin{cases} 1 & \text{w.p. } \theta_k, \\ 0 & \text{otherwise.} \end{cases}$$

- *Reward*: $r_t = y_t$,
- *Prior belief*: $\theta_k \sim \text{beta}(\alpha_k, \beta_k)$.

1.2 Online Shortest Path

Given a graph $G = (V, E, v_s, v_d)$ where $v_s, v_d \in V$, we have that

- *Mean delay*: θ_e for $e \in E$,
- *Action*: $x_t = (e_1, \dots, e_M)$, a path from v_s to v_d ,

- *Observation*: $(y_{t,e_1}, \dots, y_{t,e_M})$, where $\ln(y_{t,e}) \sim N(\theta_e - \frac{\hat{\sigma}_e^2}{2}, \hat{\sigma}_e^2)$,
- *Reward*: $r_t = -\sum_{e \in x_t} y_{t,e}$,
- *Prior belief*: $\ln(\theta_e) \sim N(\mu_e, \sigma_e^2)$.

Remark. In this model, the number of actions is equal to the number of paths from v_s to v_t and even in small graphs it can be very large and hence, it'd be intractable to search through all the paths. For instance, consider the binomial bridge shown in Figure 2.

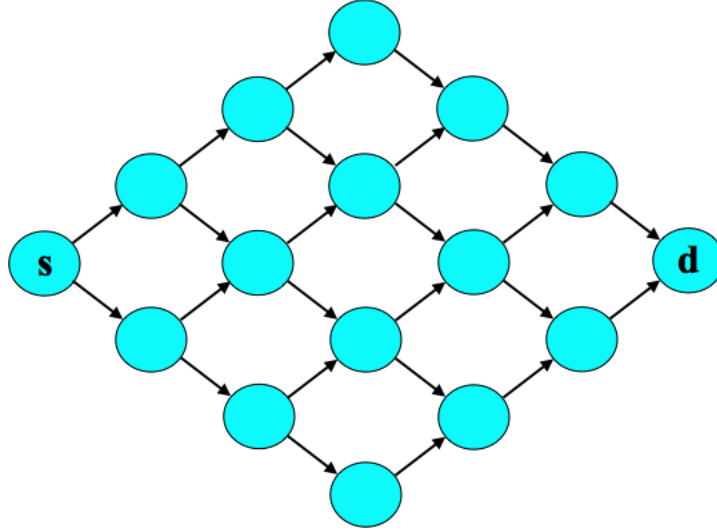


Figure 2: Binomial bridge with six stages.

2 Algorithms

In this section, we describe three commonly used algorithms.

- *Greedy*: By H_t , we denote the data observed up to the time t . Then, this algorithm works as follows:
 - $\hat{\theta} = E[\theta | H_{t-1}]$, i.e. the posterior mean of θ ,
 - $x_t = \arg \max_k \hat{\theta}_k$.

Note that the expectation is taken with respect to the posterior distribution of θ given H_t .

- *Thompson sampling*: In this algorithm, we sample $\hat{\theta}$ from the posterior distribution, instead of computing the posterior mean:
 - $\hat{\theta}$ is sampled from $P(\theta_k | H_{t-1})$ (for instance, using Gibbs sampling),
 - $x_t = \arg \max_k \hat{\theta}_k$.
- *ε -greedy*:
 - $\hat{\theta} = E[\theta | H_{t-1}]$,
 - $x_t = \begin{cases} \arg \max_k \hat{\theta}_k & \text{w.p. } 1 - \varepsilon \\ \text{unif}(\{1, \dots, K\}) & \text{otherwise.} \end{cases}$

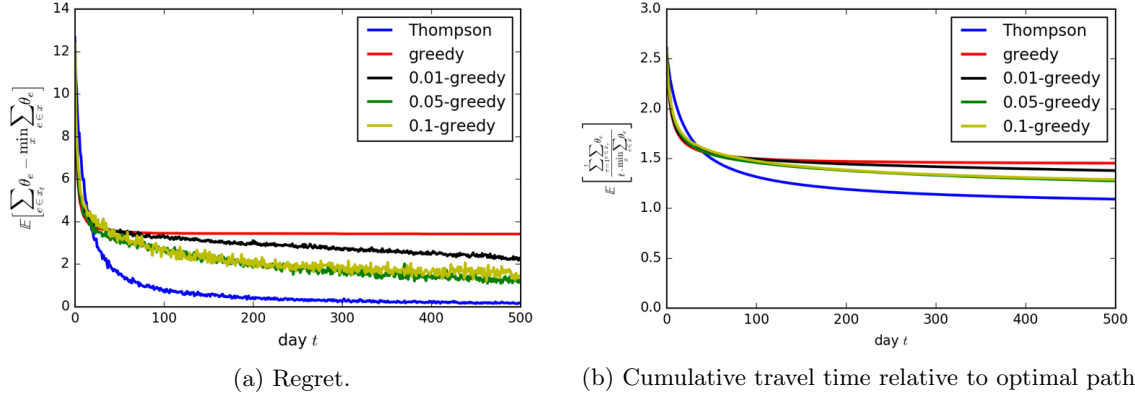


Figure 3: Comparing these algorithms for online shortest path problem.

3 Experiments

We apply these three algorithms to the online shortest path problem. For the ε -greedy algorithm, we use $\varepsilon = 0.01, 0.05, 0.1$. Then, we plot the regret of these algorithms, i.e. $E[\sum_{e \in x_t} \theta_e - \min_x \sum_{e \in x} \theta_e]$. Since, it is not feasible to compute this expectation theoretically, we generate lots of samples and compute the sample mean instead of the population mean. The result is shown in Figure 3a. As it can be seen, The Thompson algorithm has the fastest convergence rate to zero.

The other measure that we compute is *cumulative travel time relative to optimal path* given by

$$E \left[\frac{\sum_{\tau=1}^t \sum_{e \in x_\tau} \theta_e}{t \cdot \min_x \sum_{e \in x} \theta_e} \right].$$

This quantity is obviously bounded below by 1. So, we search for algorithms with closest value to 1. In Figure 3b, the sample mean of this quantity is shown. Again, this figure implies that Thompson algorithm outperforms other described methods. Moreover, it seems that, even though the greedy algorithm doesn't get closer to 1, but ε -greedy algorithms does, however, with slower pace.

The other question that needs more investigation is about robustness of these algorithms when the assumptions change. We change the model and we add factors that contribute to all or more than one edges of the graph. For instance, bad weather is a noise that contributes to all edges almost equally. Now, we can ask how these algorithms would work under the new model. As Figure 4 suggests, even when we consider the simpler model, Thompson algorithm performs very well.

4 Alternative Algorithms

4.1 Dynamic Programming

In terms of mathematical optimization, dynamic programming usually refers to simplifying a decision by breaking it down into a sequence of decision steps over time. This is done by defining a sequence of:

- State: Encoding of posterior (what we know about the state of the system).
- Policy: A policy is a rule (or function) that determines a decision given the available information in state S_t . Sequence of functions $\mu = (\mu_1, \mu_2, \dots)$ for choosing an action $x_t = \mu_t(S_t)$. At each time t , we keep the probability of being at each step and μ_t picks the action that is most likely to improve the reward most and then, we can update probabilities, according to the feedback the algorithm gets.

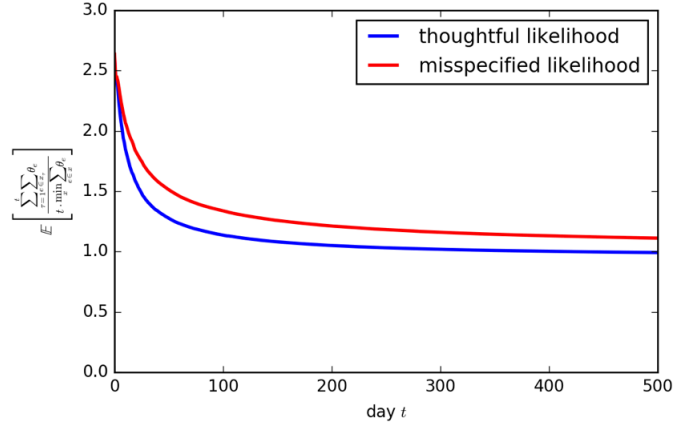


Figure 4: Performance of Thompson algorithm under true model and simpler model.

- Objective: Maximizing the reward: $E \left[\sum_{t=1}^T r_t | \mu \right]$ or $E \left[\sum_{t=1}^{\infty} \alpha^t r_t | \mu \right]$.

The problem with this is the curse of dimensionality for optimization. More precisely, let's our states correspond to an encoding of the posterior distribution. For instance, in Bernoulli bandit problem, the states of the dynamic programming consist of $2 \times K$ vectors of α_k 's and β_k 's. And even after discretizing the state space, we will have a very huge number of possibilities and we can not update all these states in a reasonable amount of time. This problem is known as *curse of dimensionality* and dynamic programming suffers from this problem.

In some situations, we can overcome this problem by using a technique called *Gittins index*. Basically, the idea is that in the problems that performing each action informs us about disjoint sets of parameters, we can decompose the above-mentioned state space and update them in an efficient way and then, the curse of dimensionality is resolved in this way. For instance, in Bernoulli bandit problem, each action only informs us about the parameters corresponding to that action and clearly these parameters are disjoint. On the other hand, in the online shortest path problem this is not the case. Because choosing one path enables us to know more about each edge in that path and so, it will affect our knowledge about other paths. Thus, we can not apply Gittins index in this case, and we need a different approach to deal with this problem.

4.2 Upper-Confidence Bound(UCB) Methods

Each method in the following sections studies an upper confidence bound (UCB) algorithm. Such an algorithm forms an optimistic estimate of the mean reward value for each action, taking it to be the highest statistically plausible value. It then selects an action that maximizes among these optimistic estimates. Figure 5 illustrates a situation that UCB algorithm chooses a different action than the one that greedy algorithm does. Greedy algorithm picks the largest parameter, so chooses action 3. However, UCB algorithm selects the action that has the highest upper confidence bound, i.e. action 1 in this case.

4.2.1 UCB for Bernoulli bandits problems

Bernoulli bandit problems are an example set of problems which can be solved by UCB methods: For each arm k ,

$$\hat{\theta}_k = E [\theta_k | H_{t-1}] + \beta \sqrt{\text{var} [\theta_k | H_{t-1}] \log t}.$$

After computing $\hat{\theta}_k$, the arm with highest $\hat{\theta}$ is played. Note that, in the above equation, the variable beta (β) is a design parameter that needs to be optimized for different set ups.

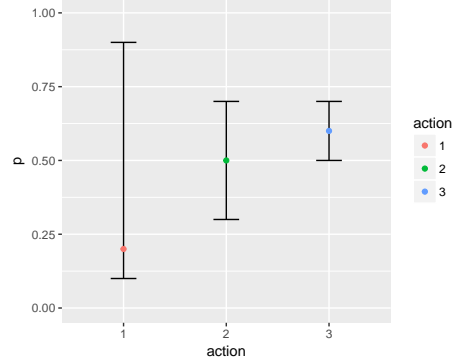


Figure 5: Estimated parameters with confidence intervals in UCB method.

4.2.2 UCB for more general problems

Θ = set of statistically plausible estimates (level set of posterior) e.g. for bernoulli $\Theta = \{\hat{\theta} : \underline{\theta}_k \leq \hat{\theta}_k \leq \bar{\theta}_k, \forall k\}$
 $x_t = \arg \max_x U_t(x)$

where: $U_t(x) = \max_{\theta \in \Theta} E[r_t | H_{t-1}, \theta = \hat{\theta}, x_t = x]$

Note that the maximization step in UCB could lead to an NP-hard problem in many cases.

4.2.3 Comparison between UCB method and Thompson algorithm

Even though it is commonly said that Thompson algorithm outperforms UCB method, but this is not the case. However, UCB method has free parameters and it requires tuning and it is usually difficult to do that, but if the parameters are set carefully, it performs better than Thompson algorithm. Figure 6 shows the comparison between these methods after choosing different parameters for UCB method.

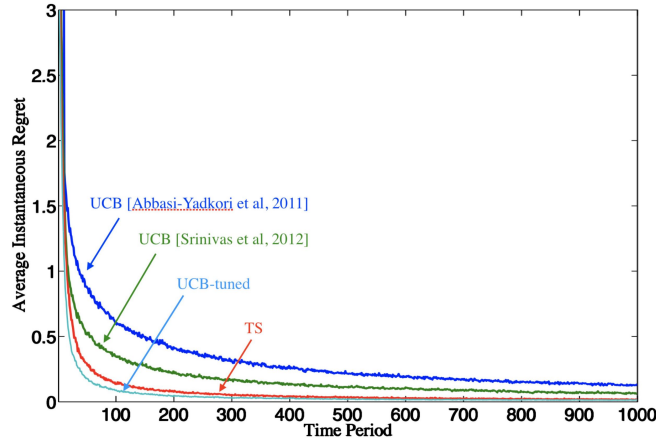


Figure 6: Comparison between UCB and Thompson.

5 Linear bandit problem (LBP)

The linear bandit problem is an extension of the classical multi-armed bandit problem that have a linear reward functions. The bandit problem is maybe the most generic way to model an exploitation-exploration trade-off. The problem can be defines as a gambler at a row of slot machines with different *parameters*, deciding which arms to play. As he plays each arm and machine, the machines provides a random *reward* from a *probability* distribution specific to that machine. The objective of the gambler is to maximize the sum of rewards earned from a sequence of *actions* (pulling the arms). The mathematical formulation of the problem is as follow:

- Action: $x_t \in \mathbb{R}^d, A_{x_t} \leq b$
- Parameters: $\theta \in \mathbb{R}^d$
- Reward: $r_t = y_t = \theta^T x_t + w_t, w_t \sim N(\mu_t, \Sigma_t)$
- Prior: $\theta \sim N(\mu_0, \Sigma_0)$, posterior $\sim N(\mu_t, \Sigma_t)$

5.1 Thompson Sampling for LBP

Many strategies exist which provide an approximate solution to the bandit problem. The idea is that the number of pulls for a given lever should match its actual probability of being the optimal lever. Thompson sampling is a probability matching strategy to sample from the posterior for the mean value of each arm. The following, shows the steps of Thompson sampling method to solve the LBP problem:

1. Sample $\hat{\theta} \sim N(\mu_{t-1}, \Sigma_{t-1})$
2. Optimize $x_t \leftarrow \max_{s.t. Ax \leq b} \hat{\theta}^T x$.

5.2 UCB for LBP

One of many strategies to solve the LBP problem is UCB algorithm. USB assumes a linear dependency between the expected reward of an action and its context. It solves LBP by modeling the representation space using a set of linear predictors. The following, shows the steps of USB method to solve the LBP problem:

1. Define confidence set $\Theta = \{\theta \in \mathbb{R}^d : (\theta - \mu_{t-1})^T \Sigma_{t-1} (\theta - \mu_{t-1}) \leq \beta \log t\}$
2. Optimize $x_t \leftarrow \max_{s.t. Ax \leq b} \max_{\hat{\theta} \in \Theta} \hat{\theta}^T x$

As mentioned previously in section 4.2.2 the above optimization problem is a NP-hard problem.