MS&E338 Reinforcement Learning Lecture 1 - Jan 09 2017

Introduction to Bandit Problems

Lecturer: Ben Van Roy *Scribe: Ahmadreza Momeni, Abubakar Abid*

1 How to think about bandit problems

Bandit problems refer to situations in which an agent is trying to maximize a reward function by choosing actions strategically as it learns about the world. One way to think about bandit problems is that it is a *generalization* of supervised learning. In supervised learning, the algorithm has access to a round (or potentially multiple rounds) of observed input features and labels. We can denote these as:

$$(\boldsymbol{x}_1, \boldsymbol{y}_1), (\boldsymbol{x}_2, \boldsymbol{y}_2), \ldots, (\boldsymbol{x}_T, \boldsymbol{y}_T),$$

in which case, each $\boldsymbol{x}_t$ is the vector of input features observed in the tth round, and $\boldsymbol{y}_t$ is the corresponding label. By using traditional techniques from supervised learning, such as regression, we can come up with a function $f \in \mathcal{F}$ that maps input features to output labels in a way that the error rate in guessing label is minimized (in fact, it is approximately minimized with high probability but we do not intend to get into the details).

Bandit problems can be seen as a generalization of supervised learning, where we:

- choose $\boldsymbol{x}_t$, the input features (or hereby known as input *actions*),
- observe some response y_t,
- receive the instantaneous reward $r_t = r(y_t)$,
- and the ultimate goal is to collect as much reward as possible; for example, we may want to maximize the mean cumulative reward $\mathbb{E}\sum_{t=1}^{T} r_t$

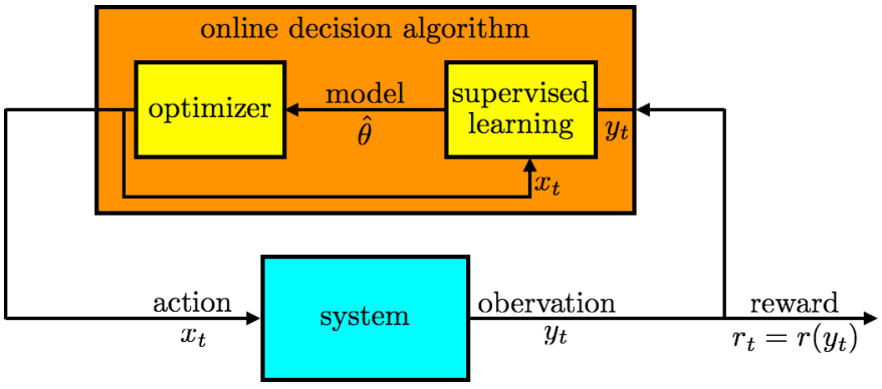

Figure 1: The general sketch of bandit problems

The term "bandit learning" is usually reserved for problems where the reward is the sole feedback. The broader class of problems, where there can be feedback beyond reward (as indicated in figure 1, for example) is sometimes called online decision problems.This is a very general framework for thinking about bandit problems, and in specific examples, there might be slight differences in the way that a problem is formulated. So let's get started and dive into some specific instances of these types of problems.

2 The Bernoulli bandit

Perhaps the simplest problem that embodies the nature of Reinforcement Learning is the classical Bernoulli bandit problem. In this problem, an agent has the option to pull one of K different arms, and each arm has a specific mean reward (probability of winning). Now the question is the following: what is the best strategy that the agent can employ in pulling the arms in order to maximize the expected amount of reward collected? To see the challenge here, note that in the beginning we have no information about the mean rewards of different arms and we need to learn about them throughout the game. On the other hand, at each time step, we have two choices. Either we can choose the arm that has the highest mean reward based on the observations so far, or we can aim to collect more information about the mean rewards by playing the arms that seem not to be the best options based on the past. In literature, it is known as the *exploration-exploitation* trade-off, which we need to address here.

Let us introduce a few notations:

- $\mathcal{X} = \{1, 2, \ldots, K\}$ is the set of possible actions.
- $\boldsymbol{\theta} = \big(\theta_1, \theta_2, \ldots, \theta_K\big)$ is the vector of the mean rewards.
- $\hat{\boldsymbol{\theta}}_t = \big(\hat{\theta}_1, \hat{\theta}_2, \ldots \hat{\theta}_K\big)$ is the conditional expected mean reward vector based on our prior information and the observations up to time t.
- $y_t \in \{0, 1\}$ is the observation at time t.
- As such, the natural reward function is $r(y_t) = y_t$.

How do we go about maximizing the expected total reward?

3 Approaches to bandit problems

There are different strategies that one could take to solve this problem. In this lecture, we discuss two of them very briefly.

3.1 Greedy algorithm

Let π denote the prior distribution of the mean rewards vector $\boldsymbol{\theta}$, and $\mathcal{H}_t = \big(x_1, y_1, \ldots, x_t, y_t\big)$ denote the history of the system up to time t. Also, let $\pi(\boldsymbol{\theta}|\mathcal{H}_{t-1}) \propto \pi(\boldsymbol{\theta}) \cdot \mathbb{P}(\mathcal{H}_{t-1}|\boldsymbol{\theta})$ denote the posterior of $\boldsymbol{\theta}$ at time t. The greedy algorithm suggests computing the posterior mean rewards vector and then choosing the arm corresponding to the highest posterior mean reward. In other words

$$\hat{\boldsymbol{\theta}}_t = \mathbb{E}[\boldsymbol{\theta}|\mathcal{H}_{t-1}], \tag{1}$$

$$x_t = \arg\max_k \Big(\hat{\boldsymbol{\theta}}_t\Big)_k. \tag{2}$$

As the final note, a typical prior that is used for these kinds of algorithm is $U(0, 1)^K$.

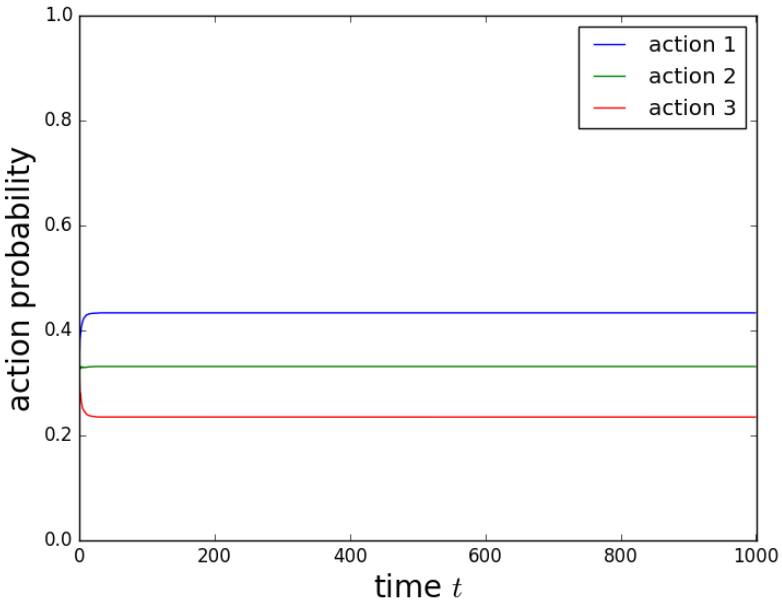

Figure 2: An experiment on the greedy algorithm with $K = 3$

3.2 Thompson sampling algorithm

Knowing the greedy algorithm, it becomes much simpler to explain Thompson sampling algorithm. In fact, these two algorithms are the same except for the estimation $\hat{\boldsymbol{\theta}}_t$ that they use. More explicitly, in Thompson sampling, instead of working with the posterior expected value of the mean rewards, we sample $\hat{\boldsymbol{\theta}}_t$ from the posterior distribution and then similar to the greedy algorithm, choose the arm corresponding to the highest estimated mean reward. In other words

$$\hat{\boldsymbol{\theta}}_t \sim \pi(.|\mathcal{H}_{t-1}), \tag{3}$$

$$x_t = \arg\max_k \left(\hat{\boldsymbol{\theta}}_t\right)_k. \tag{4}$$

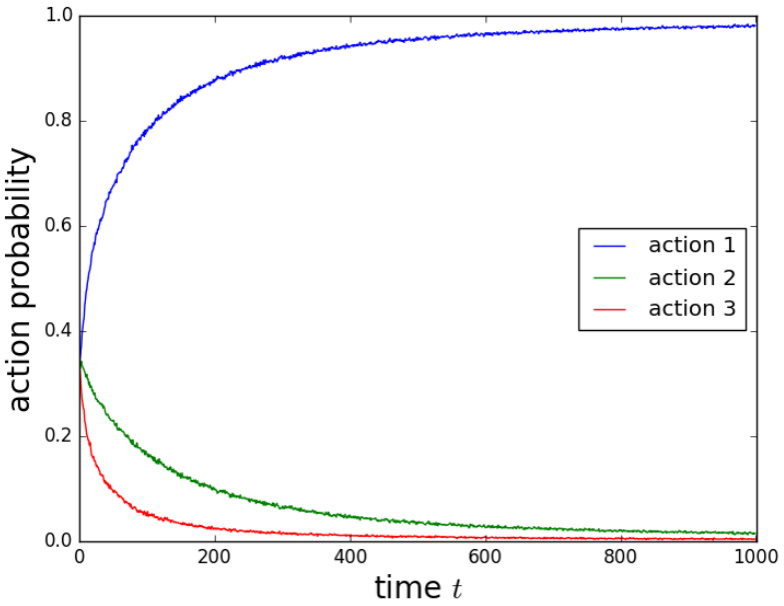

Figure 3: An experiment on the Thompson sampling algorithm with $K = 3$

3.3 Thompson sampling vs the greedy algorithm

As we saw earlier, these two algorithms are different only in the posterior belief (posterior estimation of the mean rewards) that they use. However, this tiny difference in estimation results in a huge difference in their performances. Figure 2 and 3 display the performance of these two algorithms in the same bandit settings, where the first and the third action are the best and the worst action, respectively. We note that the probability of choosing the best action tends to one if we use Thompson sampling algorithm while it doesn't if we use the greedy algorithm. Intuitively, this is due to the fact that Thompson sampling algorithm doesn't only exploit the collected information but also does explore for more information where such a behavior is absent when we use the greedy algorithm. Apart from the probability of choosing different arms, we can also compare these two algorithms in terms of their unconditional and conditional instantaneous regret defined as $\mathbb{E}\left[\max_k \theta_k - \theta_{x_t} \,|\, \theta\right]$ and $\mathbb{E}\left[\max_k \theta_k - \theta_{x_t}\right]$, respectively (note that the unconditional instantaneous regret is defined with respect to a given prior over $\boldsymbol{\theta}$). As figures 4, and 5 witness, Thompson sampling algorithm learns about the world and uses its knowledge very well since both mentioned quantities tend to zero when this algorithm is employed while they don't if we use the greedy algorithm.

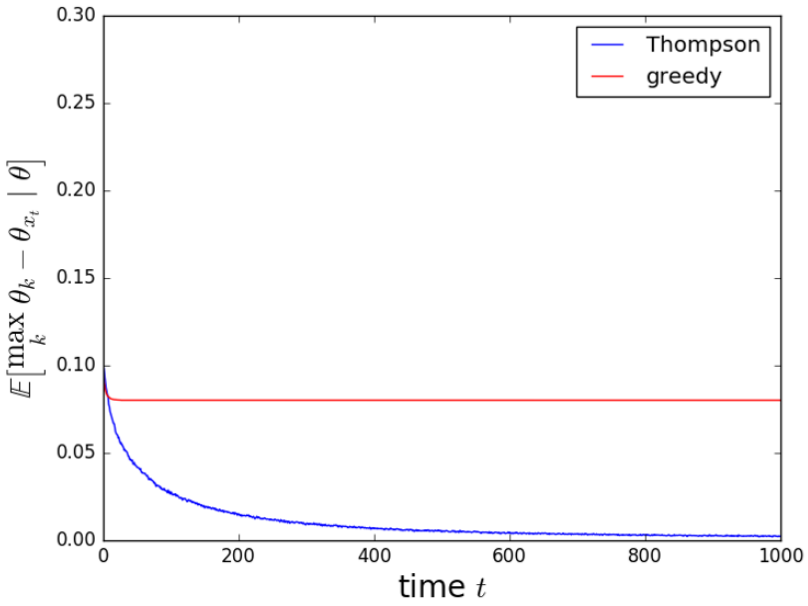

Figure 4: Comparing the conditional instantaneous regret of Thompson sampling and the greedy algorithm

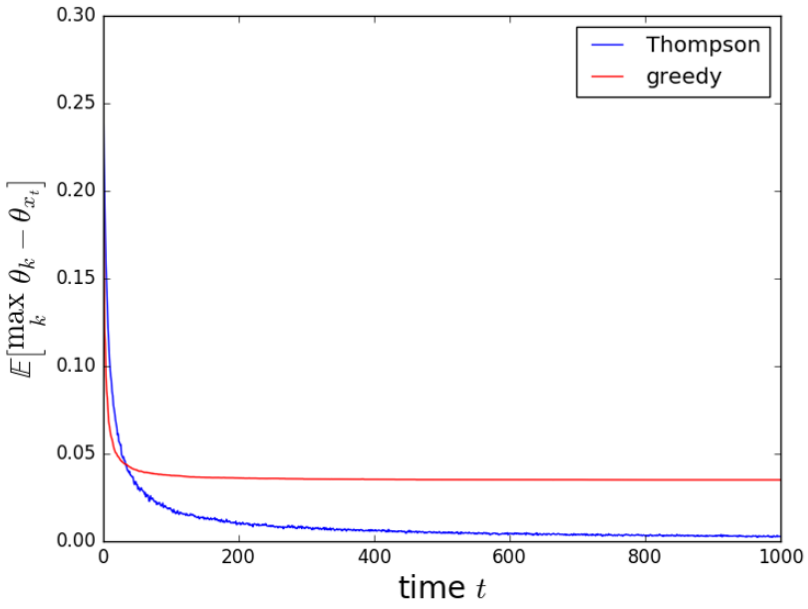

Figure 5: Comparing the unconditional instantaneous regret of Thompson sampling and the greedy algorithm

3.4 Computing posteriors with Bernoulli bandit

Both the greedy approach and Thompson sampling require computation of posterior distributions. In the case of the Bernoulli bandit, the posterior distributions take a particularly simple form, when expressed using Beta distributions. In particular, the prior, which is assumed to be a uniform distribution between zero and one, is $\text{Beta}(1,1)$. If n wins and m losses are observed from an arm, then by Bayesian rule, its posterior distribution gets updated as $\text{Beta}(n+1, m+1)$. As a property of Beta distribution, if $X \sim \text{Beta}(\alpha, \beta)$, then $\mathbb{E}X = \frac{\alpha}{\alpha+\beta}$, which is useful if we are dealing with the greedy algorithm.

If we are employing Thompson sampling and we need to sample $\hat{\theta} \sim \text{Beta}(n+1, m+1)$, one possible solution is to generate random numbers $a_i \sim \text{Exp}(1)$, $i = 1, \ldots, n$, and $b_j \sim \text{Exp}(1)$, $j = 1, \ldots, m$, then one can show that the fraction $\frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} a_i + \sum_{j=1}^{n} b_i}$ is distributed according to $\text{Beta}(n+1, m+1)$.

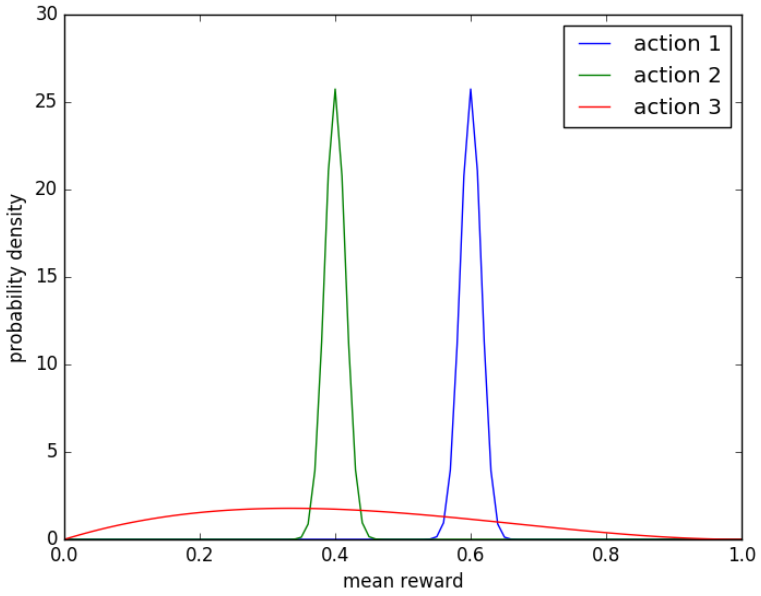

Figure 6: An example of the posterior distribution of the arms

4 Beyond Bernoulli bandit: shortest path problem

Consider a graph $G = (V, E)$ with a source and destination node, in which we want to travel from the source node to the destination node, and in this transition we may experience some delay which is the sum of the random delays experienced at each edge that we have crossed throughout the path. We can model this problem as a Reinforcement Learning problem in which the set of possible paths from the source to the destination is the set of actions, the delays observed at different edges along the path is the observation and the sum of the delays is the loss that we incur. As we can notice, there is a crucial difference here with the setting in the bandit problem, that is, here we have a much richer observation since instead of just observing the summation of the delays we get to observe each single delay of the edges. Also, the observation associated with one path may provide us with some information about the other paths that share edges with that path.

Apart from richer observations, we may deal with some problems in which actions can have delayed consequences. These type of problems are known as online decision problems (or online learning problems

or online optimization problems, ...).

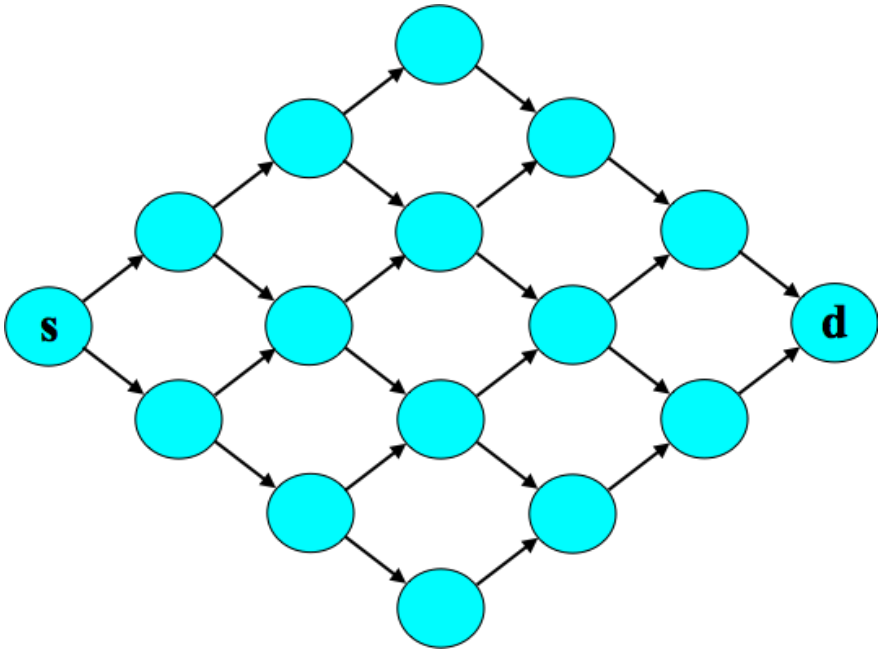

Figure 7: An example of the shortest path problem graph

5 A generalization of bandit problems: Reinforcement Learning

In this section, we are going to briefly explain how bandit problems are related to Reinforcement Learning problems. In this regard, consider the system that we discussed in the first section, and assume this system has some state s_t that evolves through time. In this case, we are not facing a bandit problem anymore, and the problem is recognized as an RL problem. To be more concrete, an RL problem consists the following parts:

- $\boldsymbol{x}_t$ is the action taken by the player at time t.
- s_t is the state of the system at time t.
- a response y_t is observed which is dependent on the action and the state of the system.
- an instantaneous reward $r_t = r(y_t, s_t)$ is received at time t.
- the next state s_{t+1} is realized dependent on x_t and s_t.
- Similar to bandit problems, the ultimate goal is to collect as much reward as possible.

It is worth mentioning that the presence of a state in the problem gives rise to delayed consequences of actions. In other words, since the action that we take at some time step can influence the later states of the system, then this action's effect can spread throughout the future.