Optimal Learning for Multi-Armed Bandits

Matthew Darlington

STOR-i | Lancaster University

What is a Multi-Armed Bandit?

- The term multi-armed bandit comes from thinking of a gambler sitting in front of a row of slot machines and deciding which they wish to play.

- I am only focusing on Bernoulli multi-armed bandits, which can be thought of as tossing a weighted coin and receiving £1 for heads, or nothing for tails.
- The results we have observed from the bandits are stored in two vectors at each time step, one to count successes $\boldsymbol{\alpha}^t$ and one for failures $\boldsymbol{\beta}^t$. We can then estimate of the parameters of each bandit at time t by:

$$p_i^t = \mathbb{E}\left[p_i \;\middle|\; S^t\right] = \frac{\alpha_i^t}{\alpha_i^t + \beta_i^t}$$

- The objective of the problem is to maximise the total rewards received before some predetermined time limit $\mathcal{H}$, called the horizon.
- The main dilemma is to balance exploiting what we believe to be the best choice at this time, with exploring to find more information out about the system.

Dynamic Programming

- We can consider the problem as a Markov decision process, and use dynamic programming to calculate the optimal policy using Bellman's equation:

$$\mathbb{V}(S^t, t) = \max_{i \in B}\left[r(S^t, i) + a\sum_{S^{t+1}} \mathbb{P}(S^{t+1}|S^t, i)\mathbb{V}(S^{t+1}, t+1)\right]$$
$$\mathbb{V}(S^{\mathcal{H}}, \mathcal{H}) = 0$$

- We can then substitute in the terms for our problem to get:

$$\mathbb{V}(S^t, t) = \max_{i \in B}\left\{p_i^t\left[1 + a\mathbb{V}(S_+^t, t+1))\right] + (1 - p_i^t)a\mathbb{V}(S_-^t, t+1)\right\}$$
$$\mathbb{V}(S^{\mathcal{H}}, \mathcal{H}) = 0$$

  Where $S_{+/-}^t$ denotes if we observe a success or failure from bandit i from state S^t respectively.

- Although this is the optimal solution, by design it is highly time and memory consuming meaning it is physically impossible to use for large horizons.
- Thus this gives motivation to have a heuristic that can be calculated in a suitable time for large problems.

Greedy Policies

- The simplest idea is to be greedy, in the sense that we always pick the arm that we believe to offer the best rewards. The problem with never exploring is that we may never even try to pull the optimal arm depending on the results from the others.
- A simple improvement to the greedy policy is to introduce a chance of randomly pulling any arm. This allows us to gain information about the other arms which will help to determine which is the best choice, for us then to be greedy with.
- The best way to do this is to use a decreasing sequence ϵ_t giving it the name epsilon decay greedy.
- One of the problems with this method is that we have to change these parameters to be optimal for different scenarios such as the length of the horizon, or number of bandits.

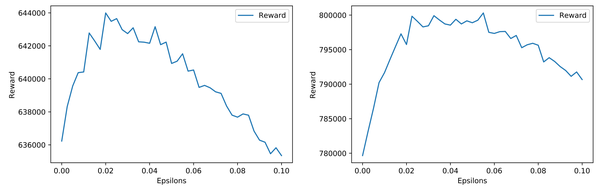

Fig. 2: Looking for the optimal parameter for epsilon greedy for 2 and 5 bandits over 1000 simulations and $\mathcal{H} = 1000$

- We can see how with more bandits we should use a higher value of ϵ. What we would expect as with more information available we would want to spend longer to find out about the system.

Thompson Sampling

- Each arm is given a Beta(α_i^t, β_i^t) prior distribution. We take a sample from these and pull the arm with the largest. The posterior distribution is then Beta$(\alpha_i^{t+1}, \beta_i^{t+1})$ due to conjugacy properties of the distributions.
- Thus the more likely we believe the arm to be optimal, the more likely we are to choose it. The advantage is the policy will not get 'stuck' with an inferior arm like a greedy method could, and given an infinite horizon will eventually find the best arm.

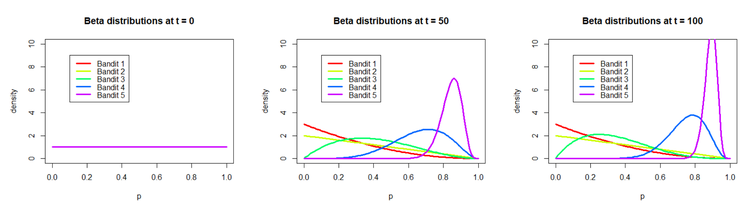

Fig. 3: Thompson Sampling on $(p_1, p_2, p_3, p_4, p_5) = (0.1, 0.3, 0.5, 0.7, 0.9)$

- Here we can see how the policy quickly learns to pick the optimal arm almost every time. During the beginning though there is the opportunity to try the others just in case they have been unlucky.

Knowledge Gradient

- We want to represent the knowledge that we can gain by choosing an arm in a numerical way.
- We consider the choice we would make if we have one more decision to learn, and then must stick with our best option until the end of the horizon.
- The value of being in a state can be defined as,

$$\mathbb{V}^t(S^t) = p_j^t = \max_i p_i^t$$

- We then define the knowledge gradient as,

$$v_i^{KG,t} = \mathbb{E}\left[\mathbb{V}^{t+1}(S^{t+1}(i)) - \mathbb{V}^t(S^t) \;\middle|\; S^t\right]$$

- Our choice is then made by picking,

$$\arg\max_i p_i^t + (\mathcal{H} - t)v_i^{KG,t}$$

- We can then substitute in our values for these and come up with a decision determined by just the values of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$.
- The policy behaves very well when we have few successes, but badly for high. This is due to the S curve effect caused by the low value of information.

Results

I have chosen to compare the methods on 2 bandits with probabilities drawn uniformly from [0,1]. We can change the scenario and see different policies have different responses, but this is the fairest for an overall comparison

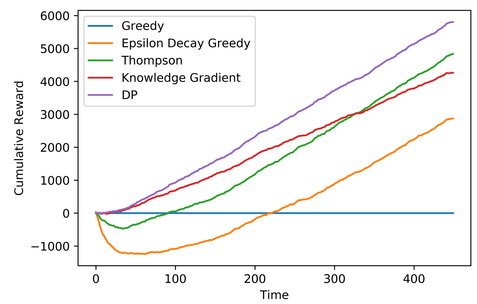

Fig. 4: Comparing the cumulative reward for each of the policies for 1000 simulations and $\mathcal{H} = 450$

- At the beginning the epsilon decay greedy performs the worst as it has a high proportion of purely random decisions.
- Thompson Sampling also takes a while to find good choices, as we would expect looking at what the posterior distributions are.
- Knowledge gradient starts off well, but can't keep up with the other methods. In some scenarios the method performs very poorly which pulls down the overall performance.
- As we would expect dynamic programming is significantly better than all the heuristics.