The AI-ML and Data Science Club of IITH

Algorithms to Solve Multi-Arm Bandit Problem-I

Contents

Compiled by : `Sriram Gonella`

1 Multi-Arm Bandit: Setting

First, we will try to establish the bandit setting. Let us consider a multi-arm bandit with **K** arms, and you are provided with **N** rounds. You are allowed to pull any of the **K** arms in each round **t ∈ 1, 2, 3, 4....., N**. On pulling arm a, the agent gets a random reward r_a sampled from a distribution(independent of previous choices and rewards). The goal is to find an algorithm that maximizes the sum of rewards obtained by pulling arm (in expectation).

2 Multi-Arm Bandit: Formulation

A multi-arm bandit can be defined as a tuple $< A, R >$. $< A >$ is the set of arms available. $R^a(r) = P(r|a)$ is the unknown distribution of rewards of arm a. At each step t, the agent selects an action $a_t \in A$ and gets a reward r_t R^{a_t}. The goal is to maximize cumulative reward $\Sigma_{t=1}^{N} r_t$.

3 Regret Minimization

The goal is to maximize cumulative reward $\Sigma_{t=1}^{N} r_t$. Define the action value function $Q(a)$ to be the mean reward for action a i.e. $Q(a) = E(r|a)$. The optimal value of V^* is

$$V^* = Q(a^*) = \max_{a \in A} Q(a) \tag{1}$$

The regret is the lost opportunity at one step

$$l_t = E[V^* - r_t] = V^* - E[r_t] \tag{2}$$

Total regret is the total opportunity loss

$$L_N = E[\sum_{t=1}^{N}(V^* - r_t)] = NV^* - E[\sum_{t=1}^{N} r_t] \tag{3}$$

So to achieve maximum cumulative reward, the total regret must be minimized.

4 Regret Minimization: Alternative Formulation

Let the **count** $N_t(a)$ be the number of times arm a is pulled up to time t. $(N_t(a) \equiv \sum_{T=1}^{t} 1_{A_T=a})$. Let Δ_a be the gap between the optimal reward (from optimal action a^*) and the reward of the arm a

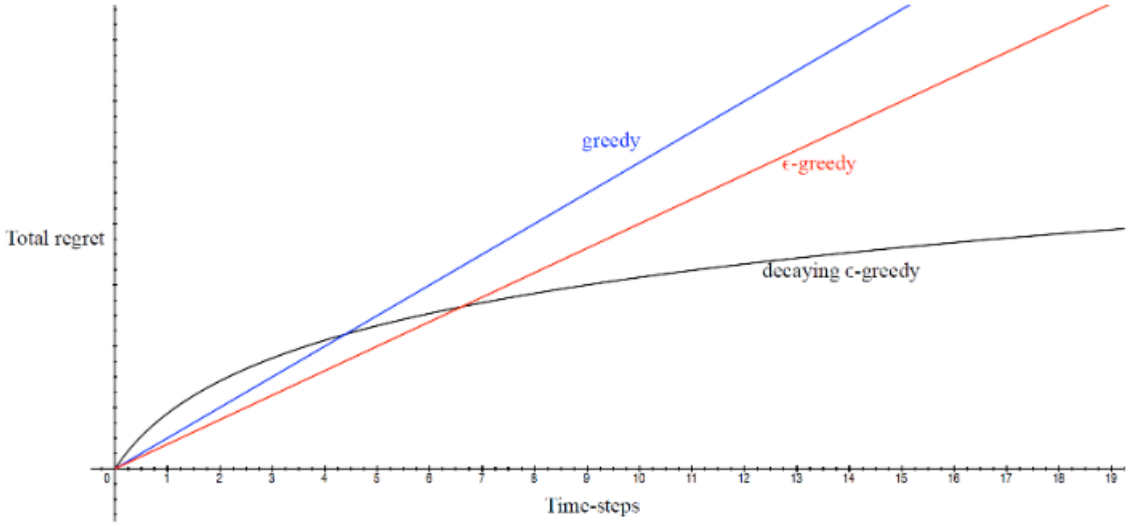
$$\Delta_a = V^* - Q(a) \tag{4}$$

Regret is the function of gaps and counts given as

$$L_N = E[\sum_{t=1}^{N}(V^* - r_t)] = E[\sum_{t=1}^{N} \Delta_{A_t}] = E[\sum_{t=1}^{N} \sum_{T=1}^{t} 1_{A_T=a} \Delta_a] = \sum_{a \in A} \Delta_a E[N_N(a)] \tag{5}$$

A good algorithm ensures small counts for large gaps.

5 Linear and Sub-Linear Regret



⇒ **Algorithms that explore forever have total linear regret**. Let's consider an algorithm that explores forever. This means that the algorithm will always choose a random action with some probability, even if it has already learned that a different action is more optimal. In other words, the algorithm will never be 100% certain about which action to take. This has the following consequences:

1. The algorithm will never achieve a perfect score. Even if the algorithm has learned the optimal policy, it will still make mistakes due to exploration.
2. The algorithm's regret will grow linearly with time. This is because the algorithm will continue to make mistakes, and each mistake will add to its regret.

To prove this, let's consider a simple example. Suppose there are two actions, A and B, and the optimal policy is to always choose action A. Let's also suppose that the algorithm explores with probability ϵ.

The expected regret of the algorithm after the T steps is then given by:

$$E[R_T] \equiv T \times \epsilon \times (R_A - R_B) \tag{6}$$

where R_A is the reward for action A and R_B is the reward for action B. We can see that the expected regret is linear in T. This is because the algorithm will explore with probability ϵ on each step, and each exploration will add to its regret. This result holds for any algorithm that explores forever. In other words, any algorithm that explores forever will have total linear regret.

⇒ **Algorithms that never explore have total linear regret.** Let's consider an algorithm that never explores. This means that the algorithm will always choose the same action, even if it is not the optimal action. In other words, the algorithm will never learn from its mistakes.

This has the following consequences:

1. The algorithm will never achieve a perfect score. Even if the algorithm starts with the optimal action, it will never learn to adapt to changes in the environment.

2. The algorithm's regret will grow linearly with time. This is because the algorithm will continue to make the same mistake, and each mistake will add to its regret.

As considered above. Suppose there are two actions, A and B, and the optimal policy is to always choose action A. Let's also suppose that the algorithm always chooses action B.

The expected regret of the algorithm after T steps is then given by:

$$E[R_T] \equiv T \times (R_A - R_B) \tag{7}$$

We can see that the expected regret is linear in T. This is because the algorithm will always choose action B, which is suboptimal, and each choice of action B will add to its regret.

This result holds for any algorithm that never explores. In other words, any algorithm that never explores will have total linear regret.

Now let us look into the actual algorithms and their pseudo codes to solve a multi-arm bandit problem.

6 Naive Approaches

6.1 Greedy Algorithm

At any time t, a greedy algorithm selects the action with the highest $\hat{Q}_t(a)$, i.e.

$$a_t^* \equiv arg \max_{a \in A} \hat{Q}_t(a) \tag{8}$$

The greedy algorithm can lock into a sub-optimal arm forever. We have discussed earlier that the greedy algorithm has total linear regret. The agent always selects the arm that is estimated to have the highest reward at any given time step. It prioritizes the exploitation of the currently perceived best arm without any exploration, leading to potential limitations in finding the truly optimal arm.

To understand how the completely greedy algorithm works, let's consider an example scenario. Suppose you have three slot machines in a casino, labeled $A, B, andC$. Each machine has an unknown probability distribution of winning, and your goal is to maximize your total winnings over a certain number of rounds.

Initially, since you have no information about the machines, you randomly select one of the arms to start. Let's say you choose to arm A and receive a reward of 5. Based on this single data point, you estimate that arm A has the highest reward among the three arms.

Now, using the completely greedy algorithm, you always select arm A in subsequent rounds because it is currently perceived as the best arm. Let's say you play arm A for the next 10 rounds and receive rewards of $3, 4, 2, 5, 3, 4, 2, 4, 3, and2$. The average reward of arms A based on these rounds is $(3 + 4 + 2 + 5 + 3 + 4 + 2 + 4 + 3 + 2)/10 = 3.2$.

However, during these 10 rounds, you didn't explore other arms like $BandC$. It is possible that one of these unexplored arms could have a higher average reward than arm A. For instance, arm B might have an average reward of 4.5, and arm C might have an average reward of 3.8. But because the completely greedy algorithm only exploits the perceived best arm, it misses the opportunity to discover higher-rewarding arms.

In this example, the completely greedy algorithm suffers from a limitation known as "exploitation-only" or "greedy myopia." It prioritizes exploitation without any exploration, leading to potentially suboptimal decisions. The algorithm's estimate of the best arm can be inaccurate or biased due to a limited number of samples.

6.2 Explore then Commit Algorithm

"Explore then Commit" combines exploration and exploitation in two distinct phases. It begins with an initial exploration phase where the algorithm gathers information about the arms, followed by a commitment phase where it commits to exploiting the best arm based on the gathered information.

Let's illustrate the "Explore then Commit" algorithm with an example. Suppose you have four slot machines in a casino labeled $A, B, C, andD$. Your objective is to maximize your total winnings over a fixed number of rounds.

1. Explore Phase: In the initial exploration phase, the algorithm randomly selects arms and collects data about their rewards. It allocates a certain number of rounds for exploration. Let's say we allocate 20 rounds for exploration.

   During the exploration phase, the algorithm randomly selects arms in a round-robin fashion. It plays each arm an equal number of times to gather information about their rewards. For example, it may play each arm five times during the exploration phase.

2. Gather Information: As the algorithm plays each arm multiple times, it keeps track of the rewards obtained from each arm. It calculates the average reward for each arm based on the collected data.

3. Identify the Best Arm: After the exploration phase, the algorithm determines which arm has the highest average reward based on the gathered information. Let's say it determines that arm C has the highest average reward among the four arms.

4. Commit Phase: In the commitment phase, the algorithm solely focuses on exploiting the arm that was identified as the best during the exploration phase. It allocates the remaining rounds, after the exploration phase, to fully exploit the identified best arm.

During the commitment phase, the algorithm plays the best arm exclusively without any further exploration. It aims to maximize its rewards by exploiting the arm it believes to be the most rewarding based on the gathered information during the exploration phase.

The "Explore then Commit" algorithm strikes a balance between gathering information through exploration and maximizing rewards through exploitation. It combines the advantages of exploration to avoid prematurely committing to suboptimal arms with the benefits of exploitation to fully exploit the identified best arm.

It's worth noting that the success of the "Explore then Commit" algorithm depends on the accuracy of the information gathered during the exploration phase. If the exploration phase does not allocate enough rounds or fails to sufficiently explore the arms, the algorithm may not identify the true best arm, leading to suboptimal results during the commitment phase. Careful consideration should be given to the allocation of rounds for exploration to ensure reliable estimates of arm rewards. Now let's see the pseudo-code for this algorithm

Algorithm Explore then Commit

Let K be the number of arms; N be the total rounds; Initialize M

Exploration Phase

for $m = 1, 2, \cdots M$ **do**
  for $a = 1, 2, \cdots, K$ **do**
    Pull arm a; Observe reward r_a; Compute mean reward $\hat{Q}(a)$ for arm a;
  end for
end for

Exploitation Phase

for $t = MK + 1, \cdots, N$ **do**
  Pull the arm with the best mean reward [i.e. $a^* = \arg\max_a \hat{Q}(a)$]
end for

In the coming Reports, we will dive into the different algorithms like ϵ-Greedy algorithm, ϵ-Greedy with Decay Approach, Upper Confidence Bound (UCB1) algorithm, and finally the Thompson Sampling Algorithm.