Machine Learning for Mechanical Engineering, ME 343
Gradey Wang, Eric Darve
Winter 2019

Stanford University

ME 343: Homework Assignment 3

Total number of points = 100.

Submission instructions: submit your homework using gradescope. We expect you to upload your answers as a PDF file along with a zip file containing your code. See `Homework 3` and `Homework 3 Code` on gradescope.

You can create the PDF using any software you want. It is possible to write your homework paper using pen and paper, and take pictures using your phone. Make sure the lighting is sufficient. Create a single PDF with all the pages.

In this homework we are going to investigate algorithms for the bandit problem. The one armed bandit is a slot machine that you operate by pulling a long handle on its side. It's commonly found in casinos. After pulling, you may get a random reward. The multi-armed bandit problem is a classical reinforcement learning algorithm.

Imagine you have a row of one armed bandits. You start pulling the arms and look at the rewards you get. Each slot machine has different settings and some will give back larger rewards than others. The goal is then to figure out an algorithm that allows you to maximize your reward. Ideally, you would pull the arm of the bandit that gives you back the largest average reward. However, the challenge is that you do not know ahead of time which bandit has the best rewards. So you need to first explore a few bandits and based on some initial sampling try to determine which bandit is the most profitable. This is the problem solved by the UCB1 algorithm.

The UCB1 algorithm is presented in this paper:
`https://link.springer.com/content/pdf/10.1023/A:1013689704352.pdf`

An important theoretical measure of how good your algorithm is is the regret:

$$\mu^* n - \sum_{j=1}^{K} \mu_j \mathbb{E}[T_j(n)]$$

where μ_j is the expectation for the reward of bandit j, and μ^* is the largest reward (the one armed bandit we want to play the most); n is the number of times we have played; and $T_j(n)$ is the number of times the one-armed bandit j has been played.

So ideally $T_j(n) = 0$ unless j is the best one armed bandit in which case we want $T_j(n) = n$. Since

$$\sum_{j=1}^{K} \mathbb{E}[T_j(n)] = \mathbb{E}\Big[\sum_{j=1}^{K} T_j(n)\Big] = \mathbb{E}[n] = n$$

we have:

$$\mu^* n - \sum_{j=1}^{K} \mu_j \mathbb{E}[T_j(n)] = \sum_{j=1}^{K} (\mu^* - \mu_j) \mathbb{E}[T_j(n)] \geq 0$$

and equality with 0 is achieved with the optimal (but generally un-achievable) optimal strategy or policy where we simply play μ^* all the time.

So the regret is always a positive quantity and equals 0 if we play systematically the best one armed bandit (with the largest μ_j). One can derive theoretical **lower bounds** on the regret and show that it is bounded from below by $\Omega(\sqrt{Kn})$ (K is the number of bandits) for all algorithms or policies (where of course we assume the best bandit is not known by the algorithm when it starts). The UCB1 algorithm is one of the most popular algorithms and guarantees an upper bound on the regret of the form $O(\sqrt{Kn \ln n})$, which is quite good![1]

The UCB1 algorithm is as follows:

Algorithm 1 UCB1

```
  Play each of the $K$ one-armed bandits once, giving initial values for empirical mean payoffs $\overline{\mu}_i$ for bandit $i$
  **for** each round $t = K + 1, K + 2, \dots,$ **do**
      Let $n_i$ represent the number of times bandit $i$ was played so far.
      Play bandit $i$ maximizing $\overline{\mu}_i + \sqrt{2 \log n / n_i}$.
      Observe the reward $R_{i,n}$ and update the empirical mean $\overline{\mu}_i$ for the chosen bandit $i$.
```

We can analyze the algorithm intuitively as follows (this is not a rigorous derivation but it captures the essential features of what is happening). Denote

$$\Delta_i = \mu^* - \mu_i$$

This is the difference between the largest reward and reward i. If we play for a long time, we can expect $\overline{\mu}_i \sim \mu_i$. Assume for a moment that the algorithm is not working and we are sampling the K bandits uniformly. Then

$$\frac{\log n}{n_j} \sim \frac{\log n}{n/K} \longrightarrow 0$$

Now consider

$$\text{argmax}_j \ \overline{\mu}_j + \sqrt{2 \, \frac{\log n}{n_j}}$$

If the last term is small, we must pick $j = j^*$ (the arm with best reward). But if we keep playing $j = j^*$, $\log n / n_j$ must increase (n increases and n_j is fixed). So, at some point

$$\sqrt{2 \, \frac{\log n}{n_j}} = \Delta_j$$

When this happens the algorithm will pick j instead of the best arm j^* because

$$\overline{\mu}_j + \sqrt{2 \log n / n_j} = \overline{\mu}_j + \Delta_j \sim \mu^*$$

So roughly the algorithm is such that we play the best arm j^* about n times, and the other sub-optimal arms about

$$\mathbb{E}[T_j(n)] \sim 2 \, \frac{\log n}{\Delta_j^2}$$

[1] The lower bound means that given a policy there is at least one multi-armed bandit problem for which the regret is larger than the lower bound. The upper bound means that the regret is lower than the bound for all multi-armed bandit problems.

In the paper, the following bound is proved

$$\mathbb{E}[T_j(n)] \leq 8 \, \frac{\log n}{\Delta_j^2}$$

This algorithm is very efficient because it combines two powerful ideas:

1. Exploitation. The best arm is sampled nearly n times which is optimal.
2. Exploration. All other arms are sampled infinitely ($\log n \to \infty$) although $\log n$ is still small compared to n so that we are minimizing the regret. The fact that we sample all arms at least $\log n$ times is critical however. Assume for example that we sample j^* (best arm) a few times but get unlucky and obtain only very small rewards. We may be tempted to just drop that bandit. But of course this is an error. In our algorithm however, since all bandits are sampled an infinite number of times, $\overline{\mu}_j$ must eventually converge to μ_j. So this algorithm cannot possibly miss the best bandit. When it discovers j^*, it will start sampling that bandit and eventually $\mathbb{E}[T_{j^*}(n)] \sim n$, thereby leading to a small regret.

Problem 1: UCB1

We first ask you to explore UCB1 numerically and observe whether your results agree with the theory.

We first consider 5 bandits ($K = 5$). We consider the following cases:

(a) The rewards are normally distributed according to $N_1(1, 0.2)$, $N_2(0.3, 0.2)$, $N_3(-1, 0.2)$, $N_4(0.5, 0.2)$, $N_5(0.8, 0.2)$, for each bandit.

(b) The rewards according to $N_1(1, 5)$, $N_2(0.3, 5)$, $N_3(-1, 5)$, $N_4(0.5, 5)$, $N_5(0.8, 5)$

(c) The rewards are bimodally distributed according to:

$$R_1 \sim \begin{cases} N(20/9, 0.2), & \text{with probability } 0.9 \\ N(-10, 0.1), & \text{with probability } 0.1 \end{cases}$$

$$R_2 \sim \begin{cases} N(13/9, 0.2), & \text{with probability } 0.9 \\ N(-10, 0.1), & \text{with probability } 0.1 \end{cases}$$

$$R_3 \sim \begin{cases} N(0, 0.2), & \text{with probability } 0.9 \\ N(-10, 0.1), & \text{with probability } 0.1 \end{cases}$$

$$R_4 \sim \begin{cases} N(5/3, 0.2), & \text{with probability } 0.9 \\ N(-10, 0.1), & \text{with probability } 0.1 \end{cases}$$

$$R_4 \sim \begin{cases} N(2, 0.2), & \text{with probability } 0.9 \\ N(-10, 0.1), & \text{with probability } 0.1 \end{cases}$$

Be careful that owing to the random nature of the reward sampling, you may get different results as you rerun your code. You may want to explore these different realizations and comment as appropriate.

This Python code may be copy/pasted if you want to avoid a typo:

```
d_params = \
[[ 20/9, .2, -10, .1], # mu1, std1, mu2, std2
 [ 13/9, .2, -10, .1],
 [    0, .2, -10, .1],
 [  5/3, .2, -10, .1],
 [    2, .2, -10, .1] ]
```

In each of the cases:

1. 10 points. Plot the proportion of visits of bandits $\{1, 2, \ldots, 5\}$ vs. number of rounds,
2. 5 points. Plot the achieved reward vs. number of rounds,
3. 10 points. Comment on your results. What does the theory predict? Does it agree with your empirical findings?

Problem 2: Tree search

You are now going to work on implementing a Monte Carlo Tree Search using UCT, which is a version of UCB1 for tree searches. Basically, we go down the tree starting from the root. At each node we consider the list of children nodes. We look at each child node as a one-arm bandit and use UCB1 to select a node. Then we traverse the tree down following that node until we reach a leaf and evaluate its reward. The leaf reward must then be back-propagated up the root of the tree. This means that the average reward for a given node o in the tree is the average of all the sampled rewards for leaf nodes that are descendants of o. As before we try to identify the leaf node that has the largest reward on average.

The theoretical analysis of the rewards at each tree node is a bit more complicated than previously because as the UCT algorithm evolves we sample the children differently. This implies that the probability distribution of the rewards for a node o is not constant but in effect changes as we vary our strategy for sampling leaf nodes. However, as we converge, we should start sampling the node with the largest reward, so that asymptotically the probability distribution of rewards for a given tree node should converge to the probability distribution of its descendant leaf with the largest reward.

The UCT algorithm can be summarized as follows:

(a) At each level of the tree, choose the following bandit (tree node)

$$a = \operatorname{argmax}_j \; \overline{\mu}_j + \sqrt{2 \ln n / n_j}$$

where the notation is as before and take $\ln 0/0 = \infty$.

(b) Back-propagate the sampled reward of the leaf node up the tree, from the leaf node to the root node.

We consider a tree with 2 levels. Each tree node has 5 children nodes. There are 25 leaf nodes. The reward of each node $i \in \{1, 2, \ldots, 25\}$ is normally distributed according to

$$N\Big(\tanh\big((i - 13)/5\big),\ 0.1\Big)$$

1. 10 points. Plot the proportion of visits of nodes $\{1, 2, \ldots, 25\}$ vs. number of actions taken,

2. 5 points. Plot the proportion of visits to level 1 nodes $\{A, B, \ldots, E\}$ (these are the 5 nodes directly below the root) vs. number of actions taken,

3. 10 points. Plot the average reward of nodes $\{A, B, \ldots, E\}$ vs. number of actions taken, and

4. 10 points. Plot the value of the root node vs. the number of actions taken. In (1), be sure to label the curve corresponding to the node with the highest mean.

5. 10 points. Comment on your results. What does the theory predict? Does it agree with your empirical findings?

Problem 3: Comparison of UCT and UCB1

We want to evaluate whether UCT is advantageous. The idea is to compare UCT where we have a depth-2 tree with UCB1. For this, we take the setup from Problem 2. You should reuse your results from Problem 2 using UCT.

Consider now all the leaves from 1 to 25. We can apply UCB1 directly to all these leaves. This is like "flattening" the tree. We lose the hierarchical structure that was provided by the 2-level tree. Does this make any difference?

Implement UCB1 for this tree with 25 leaf nodes. Then answer the following questions:

1. 10 points. Plot the proportion of visits of nodes $\{1, 2, \ldots, 25\}$ vs. number of actions taken, and

2. 10 points. Plot the value of the root node vs. the number of actions taken.

3. 10 points. Qualitatively describe whether UCT or UCB1 is faster than the other. What are your findings? Can you provide an explanation for your observations?