Multi-armed bandits
AIMS Machine Learning Course 2017

Chris Nicholls

November 24, 2017

1 Multi-armed bandits

In this report we follow the lecture notes [Kan17] and book draft [Sli17].

The multi-armed bandit problem consists of an agent repeatedly playing a game with K actions, receiving reward after each action, and seeking to maximise this reward over all rounds. Explicitly, there are K actions (often called *arms*), and at each time step in the game, the agent selects an action $a_t \in \{1, \dots, K\}$ and receives a reward $r_t(a_t)$ for having played this action. Crucially, the agent only gets feedback for the played arm, and does not get feedback for any other arms.

In the following we assume that each game lasts a finite number of time steps T, and that the agent knows T. Then the total reward of the agent in a given episode is $\sum_{t=1}^{T} r_t(a_t)$.

1.1 Bandits with IID rewards

We first consider the multi-armed bandit problem in which each arm has a fixed reward distribution, and rewards from each arm are independent across time steps. The reward distributions between arms can differ. The agent has to balance exploring different arms, to learn about their distributions, and exploiting arms that they think provide greater reward.

In the following we assume that the reward distribution of each arm a is a Bernoulli distribution with mean $\mu(a)$. Thus at the beginning of each episode the means $\mu(1), \dots, \mu(K)$ for each arm are chosen and fixed for the episode. At each time step, if the agent chooses action i, they receive reward $r_t(i) \sim \text{Bernoulli}(\mu(i))$.

Since reward varies a lot for different reward distributions, it makes more sense to compare the reward of the agent relative to some other metric. A useful metric to compare against is the performance of the agent that chooses the arm with highest mean every time – of course, the mean reward of each arm isn't known to the agent so this strategy can't actually be played. Nonetheless, this difference is called the *regret* of the episode, and is defined formally as follows. Let $a_1, \dots, a_T$ be the sequence of actions the agent makes, and let $\mu(1), \dots, \mu(K)$ be the (fixed) means of the Bernoulli distributions for each arm. Then the regret of the episode is

$$R(T) = \max_a \mu(a) - \sum_{t=1}^{T} \mu(a_t),$$

which is the difference between the expected reward for choosing the best arm every time and the expected reward for having taken the actions $a_1, \dots, a_T$. We also call this the *cumulative regret*, since it is summed over all time steps.

What we are most interested in is the *expected regret*, $\mathbb{E}(R(T))$, which is well-defined for a fixed agent and fixed distribution over which we generate episodes.

Remark 1. *Note that the reward the agent actually received is $\sum_{t=1}^{T} r_t(a_t)$, which differs from the second term in the definition of regret. In fact, neither term in the regret can be computed unless the mean vector μ is known. In practice, since we generate μ in our implementations, we can compute this. Even if the mean vector is not known, we can estimate it using the rewards observed by the agent, and can thus estimate the regret.*

We model the reward distribution of each arm a as a Bernoulli distribution with mean μ_a. We considered two methods of selecting μ_a.

Randomly chosen means Firstly, we consider sampling μ_a for each arm uniformly in the interval $[0, 1]$, and then playing the entire episode with those means. We call this the *randomly chosen means setup*.

Lower bound construction Secondly, we consider the lower bound construction as in the lectures. With K arms, we choose uniformly at random among the K setups I_i, where

$$\mu_j = \begin{cases} \frac{1}{2} + \epsilon, & \text{if } j = i \\ \frac{1}{2}, & \text{if } j \neq i. \end{cases}$$

We call this the *lower bound setup*.

Thus in the lower bound setup, we randomly choose one of the arms to have reward $1/2+\epsilon$, and all other arms have reward $1/2$. It is shown in lectures that if we take $\epsilon = \Theta(\sqrt{K/T})$, then all bandit algorithms have expected cumulative regret $\Omega(\sqrt{KT})$ for this setup. In the below, I take the implied constant to equal 1, and set $\epsilon = \sqrt{K/T}$, although I always cap the mean $1/2 + \epsilon$ to be at most 1.

1.2 Adversarial bandits

We also consider the *adversarial bandit* setting, where for each time step t and each arm a, an adversary chooses the rewards $r_t(a)$. The agent then chooses an action a_t and receives reward $r_t(a_t)$. Note that the adversary chooses rewards before the agent chooses their action, which makes the problem nontrivial.

We actually restrict to the case of an oblivious randomised adversarial bandit. That the adversary is oblivious means that their choice of rewards do not depend on the agent's previous actions. The adversary may as well choose all the rewards before the agent even starts playing, thus populating a *reward table* $r_t(a)$ with an entry for each $t = 1, \ldots, T$ and arm $a \in \{1, \ldots, K\}$. We assume the adversary is randomised, meaning that the adversary fixes a distribution over the reward table before each episode, and then generates rewards according to this distribution. Thus iid rewards for each arm is one example of this.

In the adversarial setting, we use the *hindsight regret*. Given the reward table, the reward in hindsight is then

$$R_{\text{hind}}(T) = \max_a \sum_{t=1}^{T} r_t(a) - \sum_{t=1}^{T} r_t(a_t),$$

where $a_1, \ldots, a_T$ is the sequence of actions the algorithm makes, and $r_t(a_t)$ is the sequence of rewards the algorithm receives. Thus the hindsight regret is the difference between the reward of the best arm in hindsight minus the reward the algorithm actually received.

Remark 2. *We could again use the definition of regret from the iid reward scenario, but this would require computing explicitly the mean reward for each arm. In many scenarios, computing this mean analytically would be tricky. Replacing each term in $R_{\text{hind}}(T)$ by its expected value under the adversary's reward distribution yields the* foresight regret.

For the adversarial bandit setting, I investigated the following scenario, suggested in the problem description. I sampled two separate mean vectors μ^1, μ^2 in $[0, 1]^K$. Again I split into two cases: using either the uniform distribution or lower bound construction. At each time step the rewards are generated by a Bernoulli distribution with mean vector either μ^1 or μ^2, chosen as follows. We first fix a transition probability p. At time $t = 1$, we set $\mu = \mu^1$. At each subsequent time step, with probability p, the mean vector μ is selected as the other mean vector.

2 Algorithms

We have provided four scenarios to test multi-armed bandit algorithms against. The scenario is either the standard bandit problem (iid rewards), or the adversarial bandit problem. In each scenario, we use rewards

generated from a Bernoulli distribution for each arm, and we either generate the mean vectors μ uniformly at random in $[0,1]^K$ or using the lower bound construction.

We now describe the algorithms we implemented and tested against these four scenarios. For each algorithm, we assume we know the number of trials T and number of arms K beforehand.

2.1 Uniform exploration

With uniform exploration, we first sample each arm a fixed number of times, then compute the sample mean for each arm, and for all remaining time steps follow the arm with highest mean.

We showed in lectures that if we explore for N steps in total, where $N = \mathcal{O}((T/K)^{2/3}(\log T)^{1/3})$, then uniform exploration has a regret bound of $\mathbb{E}(R(T)) \leq \mathcal{O}(T^{2/3}(K \log T)^{1/3})$.

Remark 3. *Note that we explore for N steps in total, so we sample each arm N/K times in the exploration phase. Technically, I used $N = 10(T/K)^{\frac{2}{3}}(\log T)^{\frac{1}{3}}$. The factor of 10 is to ensure that the algorithm always explores all arms. One could replace this with $\max(K, (T/K)^{\frac{2}{3}}(\log T)^{\frac{1}{3}})$, which also ensures full exploration. Taking a smaller factor for N may well improve performance.*

2.2 Epsilon-greedy

The epsilon-greedy algorithm keeps track of the sample mean of each arm. It has a parameter ϵ, which can depend on the time-step t. With probability ϵ it chooses a random arm, and otherwise it picks the arm with largest sample mean.

Remark 4. *A note on implementation: we assume that each arm has mean 1 until it is played, to encourage each arm to be played at least once first. Previously I tried assuming the mean was zero, but if ϵ is small and K is large, then this can cause a long delay in trying all the arms and so regret can be high to start with.*

With $\epsilon_t = t^{-1/3}(K \log t)^{1/3}$, epsilon-greedy achieves regret bound $\mathbb{E}(R(T)) \leq \mathcal{O}(T^{2/3}(K \log T)^{1/3})$.

2.3 Successive elimination

We keep sample means for each arm, and also maintain the confidence intervals $\gamma_t(a) = \sqrt{\frac{2 \log T}{n_t(a)}}$, where $n_t(a)$ is the number of times a has been played.

We now play in phases. At the start of the first phase, mark all arms as eligible. In each phase, we play each eligible arm once that has not yet been played this sequence. At the end of the phase, we update the eligible arms. An arm is eligible if and only if its upper confidence limit, $\mu_t(a) + \gamma_t(a)$ is at least the lower confidence limit of all other arms a', i.e.

$$\mu_t(a) + \gamma_t(a) \geq \max_{a'}(\mu_t(a') - \gamma_t(a')).$$

We can simply compute the maximum lower confidence limit, and check the upper confidence limit of all arms against that.

This algorithm has no parameters, and achieves regret bound $\mathbb{E}(R(T)) = \mathcal{O}(\sqrt{KT \log T})$.

2.4 UCB1

UCB stands for upper confidence bound. The algorithm `UCB1` maintains sample means for each arm, and computes the upper confidence limit $\text{UCB}_t(a) = \mu_t(a) + \gamma_t(a)$ at time t. It picks the arm with largest $\text{UCB}_t(a)$.

UCB1 achieves regret bound $\mathbb{E}(R(T)) = \mathcal{O}(\sqrt{KT \log T})$.

2.5 Thompson sampling

Thompson sampling works as follows. We first assume a prior distribution on the reward distribution of each arm. Having taken action a_t and received reward r_t at round t, we update our reward distributions using Bayes' rule to get a posterior distribution.

We make the assumption (as in [Sli17]) that the reward distribution for each arm is from a single-parameter family, with each distribution in the family completely determined by its mean. For example, the Bernoulli distribution with parameter μ, and the normal distribution with unit variance: $\mathcal{N}(\mu, 1)$. Thus at time t we assume we have a posterior distribution over reward distributions for each arm a.

Let $H_t = (a_1, r_1, \ldots, a_{t-1}, r_{t-1})$ be the history of all actions taken and rewards observed by the algorithm up until time t. Thompson sampling chooses an action a by sampling a from the distribution $p_t(a) = \mathbb{P}(a = a^* \mid H_t)$; this is the probability that a is the best arm, given the actions and rewards seen by the algorithm so far. Since we make the single-parameter family assumption, we can equivalently sample a mean $\tilde{\mu}_a$ for each arm from the posterior, and then choose the arm with highest sample mean: $\tilde{a} = \text{argmax}_a\, \tilde{\mu}_a$. This is because

$$\begin{aligned}
\mathbb{P}(a = a^* \mid H_t) &= \int \mathbb{P}(a = a^*, \mu \mid H_t) d\mu \\
&= \int \mathbb{P}(a = a^* \mid \mu) \mathbb{P}(\mu \mid H_t) d\mu \\
&= \mathbb{E}_{\mu | H_t}(\mathbb{P}(a = a^* \mid \mu)) \\
&= \mathbb{E}_{\mu | H_t}(\mathbb{1}_{a = \text{argmax}\, \mu}).
\end{aligned}$$

Note that $\mathbb{P}(a = a^* \mid \mu)$ does not depend on H_t. The last equality follows because a is the optimal arm, given μ, if $\mu(a)$ is the maximum element of μ. Hence we can sample an arm from $\mathbb{P}(a = a^* \mid H_t)$ by sampling $\mu \mid H_t$ and then choosing $\tilde{a} = \text{argmax}_a\, \tilde{\mu}_a$ as above.

I implemented Thompson sampling using a Beta distribution. Assuming rewards either equal to 0 or 1, the posterior distribution is also a Beta distribution. I used the prior $\text{Beta}(1, 1)$, and then updated each arm when I received a reward for it. After α observations of 1s and $n - \alpha$ observations of 0s from arm a, the posterior is $\text{Beta}(\alpha + 1, n - \alpha + 1)$.

Thompson sampling achieves regret bound $R(T) = \mathcal{O}(\sqrt{KT \log T})$ in the iid rewards scenario ([Sli17], Theorem 4.12).

2.6 Exp4

As explained in [Sli17] and in lectures, we can solve the adversarial bandit problem using the idea of *experts*. We define the problem of adversarial bandits with expert advice as follows. Suppose there are N experts $e \in \mathcal{E}$, who, at each time step t, recommend an arm $a_{t,e}$. The agent sees all recommendations, chooses an arm a_t to play, and then incurs cost $c_t(a_t)$. The goal is now to minimise cost with respect to the best *expert*.

We relate this to the adversarial bandit problem with K arms by defining K experts, where expert i always recommends playing arm i.

We use the adversarial bandit algorithm called `Exp4` from [Sli17]. The adversarial algorithms are usually phrased in terms of cost rather than reward. We make the adjustment $c = 1 - r$ where c denotes the cost and r denotes the reward. This takes a reward in the range $[0, 1]$ to a cost in the range $[0, 1]$ where, instead of maximising the reward, we want to minimise the cost. Thus to use the adversarial algorithm `Exp4` we make the transformation to costs and then use the algorithm as described for costs.

```
Given a set E of experts, and parameters eps, gamma in (0,1/2)
Initialise weight w_1(a) = 1 for all arms a
For each round t:
    Compute p_t(a) = w_t(a) / sum_{a'} w_t(a')

    Draw expert e_t from p_t
    Get recommendation of action b_t from expert e_t
    With probability gamma:
        pick action a_t uniformly at random
    Else:
```

```
    pick action a_t = b_t
Observe cost c_t(a_t) for playing action a_t
Define fake costs for all experts:
    fc_t(a) = c_t(a) / p_t(a), if a = b_t
            = 0, else

Update weights w_{t+1}(a) = w_t(a) * (1-eps)^{fc_t(a)}
```

`Exp4` with parameters $\gamma \in [0, \frac{1}{2T})$ and $\epsilon = \sqrt{\log K/(3U)}$ where $U = K/(1-\gamma)$, the regret is bounded by $\mathbb{E}(R(T)) \leq 2\sqrt{3}\sqrt{TK \log K} + 1$.

3 Results

I implemented uniform exploration, epsilon-greedy, successive elimination, UCB1, Thompson sampling, and Exp4 (the generalisation of Hedge), and tried each algorithm out on all four bandit setups. I also implemented a random bandit algorithm, that just chooses randomly at each stage. As suggested, I fixed $K = 10$ arms and focused on the dependence on T.

Note that some of the algorithms require parameters, and that I didn't spend long tuning them. For example, the parameter N in the uniform exploration algorithm is only upper bounded up to a constant.

The upper bounds on expected regret for all the algorithms are of the form $\mathbb{E}(R(T)) \leq \mathcal{O}(T^\alpha \log T\sqrt{K})$ for some α, where $R(T)$ denotes the cumulative regret. The upper bounds can have slightly better dependence on $\log T$ and K, but we focus on the T^α terms here. To determine α experimentally, we plot $\log R(T)$ against $\log T$, where T. To approximate $R(T)$, I ran each experiment 40 times and averaged the results. Due to computational constraints, I chose $T = 20 \cdot 1.2^i$ for $i = 1, \ldots, 45$. For the adversarial scenario I chose $p = 0.1$ as the transition probability. The results are displayed in Figure 1.

The top two graphs in Figure 1 show the results for the iid reward scenario. The top left graph is for the lower bound means, and the top right is for the randomly chosen means. The bottom two graphs show the results for the adversarial setting; again, the bottom left graph is for the lower bound means, and the bottom right graph is for the randomly chosen means.

We expect that for the iid rewards bandit scenario, the slopes are upper bounded by $2/3$ for uniform exploration and epsilon-greedy, and by $1/2$ for successive elimination, `UCB1`, Thompson sampling, and `Exp4`. This does seem to hold experimentally for most of the algorithms (top two graphs in Figure 1). In particular, for the iid rewards with random means (top right graph), we make the following observations. The random bandit algorithm seems to have slope roughly 1, while the uniform exploration, and epsilon-greedy algorithms have slightly shallower slope, which would presumably eventually be upper bounded by $2/3$, as T increases. The slopes of the remaining algorithms become shallower as T increases, again presumably eventually being upper bounded by $1/2$.

We now discuss the lower bound construction (left two graphs). As explained in lectures and in [Sli17], we can also get upper bounds on expected regret if $\Delta(a) \coloneqq \mu(a^*) - \mu(a)$ is bounded. Indeed, cumulative regret is bounded by $\mathcal{O}(T \cdot \max_a \Delta(a))$. Our choice of $\epsilon = \sqrt{K/T}$ in the lower bound construction ensures that all algorithms get a regret bound of $\mathcal{O}(\epsilon T) = \mathcal{O}(\sqrt{KT})$. Thus we expect that all algorithms should get a slope of at most $1/2$ in the graphs (in both the adversarial and non-adversarial settings). Separately, we know in the iid setting, that the lower bound construction gives a lower bound on the slope of $1/2$. Thus we expect that the lines should all have slope $1/2$ in the top left graph. This also seems to hold in the adversarial setting, though the lower bound construction doesn't immediately imply this theoretically.

For the adversarial means, we only have an upper bound on expected regret for `Exp4`. It's interesting that some of the other algorithms, not designed for the adversarial scenario, performed comparably to `Exp4`. Notably, Thompson sampling performed best in both the random reward and adversarial random reward scenarios. `UCB1` and `Exp4` performed comparably in the random non-adversarial setting, and `UCB1` outperformed `Exp4` in the random adversarial setting.

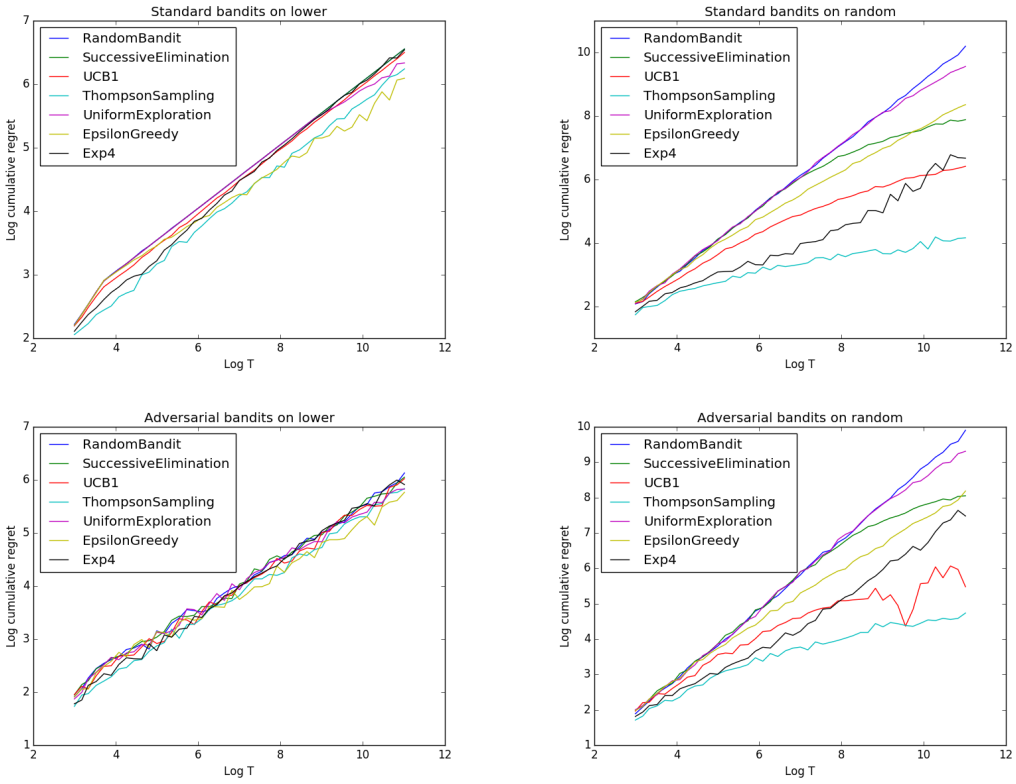

Figure 1: Log cumulative regret vs $\log T$ for the various bandit scenarios. Top left: standard setup with lower bound means; top right: standard setup with randomly chosen means; bottom left: adversarial setup with lower bound means; bottom right: adversarial setup with randomly chosen means.

4 Conclusion

In this report we compared seven multi-armed bandit algorithms on four scenarios. The consistently worst performing experimentally were the random bandit algorithm (unsurprisingly), and uniform exploration. The best performing seems to be Thompson sampling.

We confirmed experimentally, in a specific scenario, the lower bound construction from [Sli17] that posits that no algorithm can achieve better than $\Omega(\sqrt{KT})$ expected regret across all multi-armed bandit problem instances.

We also investigated experimentally the performance on adversarial bandit problems of algorithms which only have regret guarantees in the iid reward scenario. It is difficult to comprehensively test performance in the adversarial scenario, but in the scenario we tested, we found that Thompson sampling seems to adapt well to the adversarial scenario. The `UCB1` algorithm also performed well.

References

[Kan17] Varun Kanade. Machine learning lecture course (AIMS CDT, Oxford). 2017.

[Sli17] Aleksandrs Slivkins. Introduction to multi-armed bandits (lecture notes). 2017.