Multi-Armed Bandit Problem

Sunith Suresh, Lin Xiao, Ilan Man and Sanjay Hariharan

May 3, 2016

1 Introduction

This paper explores the multi-armed bandit problem (MAB). A multi-armed bandit is a sequential experiment with the goal of achieving the largest possible reward from a payoff distribution with unknown parameters. The term multi-armed bandit comes from slot machines where each machine is known as a one-armed bandit. In this set up, at each iteration the player must decide which arm of the experiment to observe next.

The task is complicated by the stochastic nature of the bandits in the following two ways:

1. A suboptimal bandit can return many winnings, purely by chance, which would make us believe that it is a very profitable bandit. Similarly, the best bandit might not yield a reward if only played a few times.
2. If we have a found a bandit that returns pretty good results, do we keep drawing from it to maintain our pretty good score, or do we try other bandits in hopes of finding an even-better bandit? How do we know when to switch and when to stick to the current bandit? This is the *exploration vs. exploitation* dilemma.

This algorithm is popular in machine learning literature as an example of reinforcement learning, as well as game theory as an example of optimal decision making in the face of multiple choices.

This paper reviews several strategies for selecting machines, including a novel (to our knowledge) approach using dynamic programming, and compares each approach against each other. Note that there are variations on the stochastic MAB, including contextual bandit and adverserial bandit, which we will not discuss here.

2 Multi-armed bandit review

In Machine Learning literature, the stochastic multi-armed bandit problem is formulated as:

> Given K machines, each with an unknown probability of yielding a reward, which come from a fixed but unknown distribution parameterized by θ_i, for $i \in (1, ..., K)$, and N total plays, decide which machines to play in order to maximize the total reward.

It is common in the literature to express the maximum reward as minimzing *regret* compared to the best arm in hindsight. That is, define regret at each step as $r_t = R^* - R_i$, where R^* is the reward yielded by selecting the best machine and R_i is the reward yielded by selecting machine i. The goal is then to devise a strategy to minimize $\sum_{t=1}^{N} r_t$. Note that since this is a stochastic problem, we aim to minimize regret in expectation or with high probability.

The common theme between the algorithms outlined in this paper are that our objective is to select the optimal machine to play at each time step, given that we want to minimize cumulative regret. The approach to selecting which machine to play is known as the *policy* or acquisition function, denoted as $U(\alpha_i, \beta_i)$. This function balances the trade off between selecting the best machine based on previous plays and the possibility of a better machine that hasn't been selected yet.

2.1 ϵ-greedy

- Sunith

2.2 Upper Confidence Bounds

The other algorithms presented in this paper are similar in that they pay attention only to how much reward they've gotten from the machines. This means that they're likely to under-explore options whose initial experiences were not rewarding, even though they may not have enough data to be confident about those arms. One naive approach to solving this problem is to run the algorithm multiple times and take the average of the results. Another could be to run the algorithm for a very long time and hope the probabilistic nature will eventually correct for random variation in the reward distribution. Another approach is to use probability theory to bound our confidence in how good or bad each machine is. This is exactly what Upper Confidence Bounds does.

The upper confidence bound (UCB) family of algorithms selects the machine with the largest upper confidence bound at each round. This paper will only focus on UCB1, but note that there are many variants of the UCB family. The more times you play a machine, the tighter the confidence bounds become. So as the number of plays for each machine increases, the uncertainty decreases, and so does the width of the confidence bound.

We want to know with high probability that the true expected payoff of a play $\hat{\mu}_i$ is less than our prescribed upper bound:

$$\bar{\mu_i} + \sqrt{\frac{2ln(t)}{n_i}}$$

Where $\bar{\mu_i}$ is the average reward obtained from machine i and n_i is the number of times machine i has been played so far.

This upper bound is the sum of two terms, where:

1. the first term is the average reward
2. the second term is related to the one-sided confidence interval for the average reward according to the Chernoff-Hoeffding bounds

Recall that since rewards follow a Bernoulli distribution, we can apply Chernoff-Hoeffding to upper bound the probability that the sum of rewards from each machine deviates from its expected value:

$$\mathrm{P}(Y + a < \mu) \leq e^{-2na^2}$$

This confidence bound grows with the total number of actions we have taken but shrinks with the number of times we have tried this particular action. This ensures each action is tried infinitely often but still balances exploration and exploitation. It can be shown that the regret for UCB1 grows with **ln n**, as witnessed below for the optimal machine.

Note that in addition to keeping track of our confidence in the estimated values of each machine, the UCB algorithm doesn't use randomness at all. Unlike the other algorithms in this paper (and in the literature) it's possible to know exactly how UCB will behave in any given situation. This can make it easier to reason about at times.

2.2.1 UCB1: Algorithm

Assuming K machines:

1. Play each arm once
2. Observe rewards r_i, for $i = 1, ..., K$
3. Set $n_i = 1$, for $i = 1, ..., K$
4. set $\bar{\mu_i} = \frac{r_i}{n_i}$
5. For time $t = K + 1, ..., N$:
   (a) Play arm $\hat{i} = \underset{i}{\operatorname{argmax}} \, (\hat{\mu_i} + \sqrt{\frac{2ln(t)}{n_i}})$

(b) Observe reward r
(c) $r_{\hat{i}} = r_{\hat{i}} + r$
(d) $n_{\hat{i}} = n_{\hat{i}} + 1$
(e) update $\hat{\mu}_i = \frac{r_{\hat{i}}}{n_{\hat{i}}}$

Below we run UCB1 on 5 machines with true reward distribution, $[0.05, 0.1, 0.3, 0.2, 0.5]$:

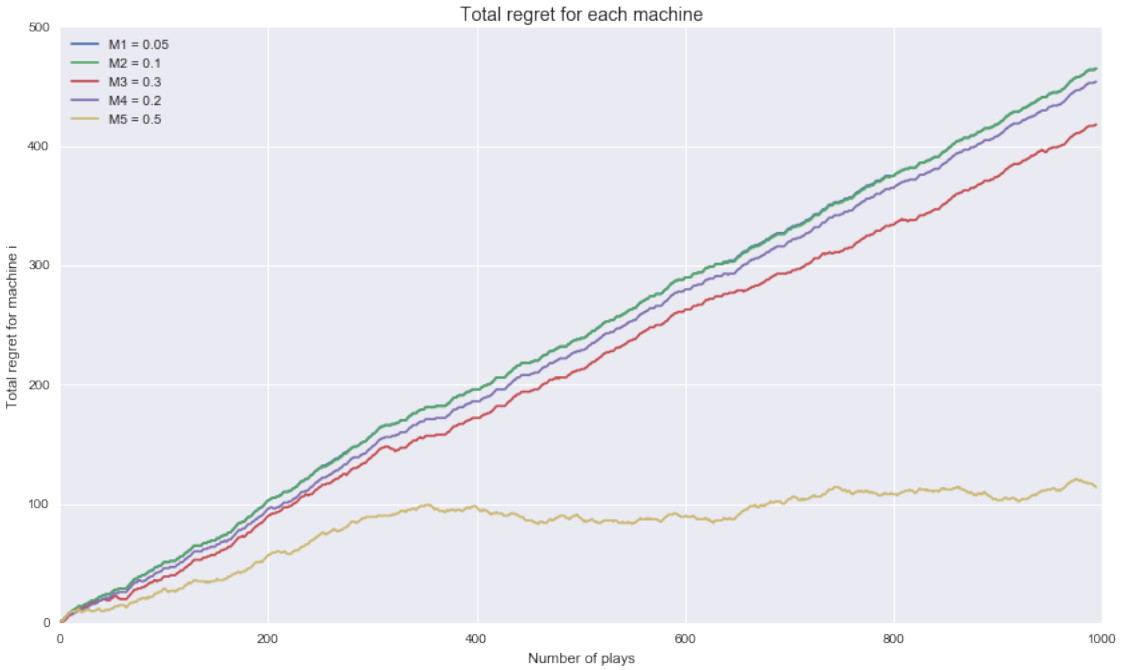

Figure 1: Cumulative Regret

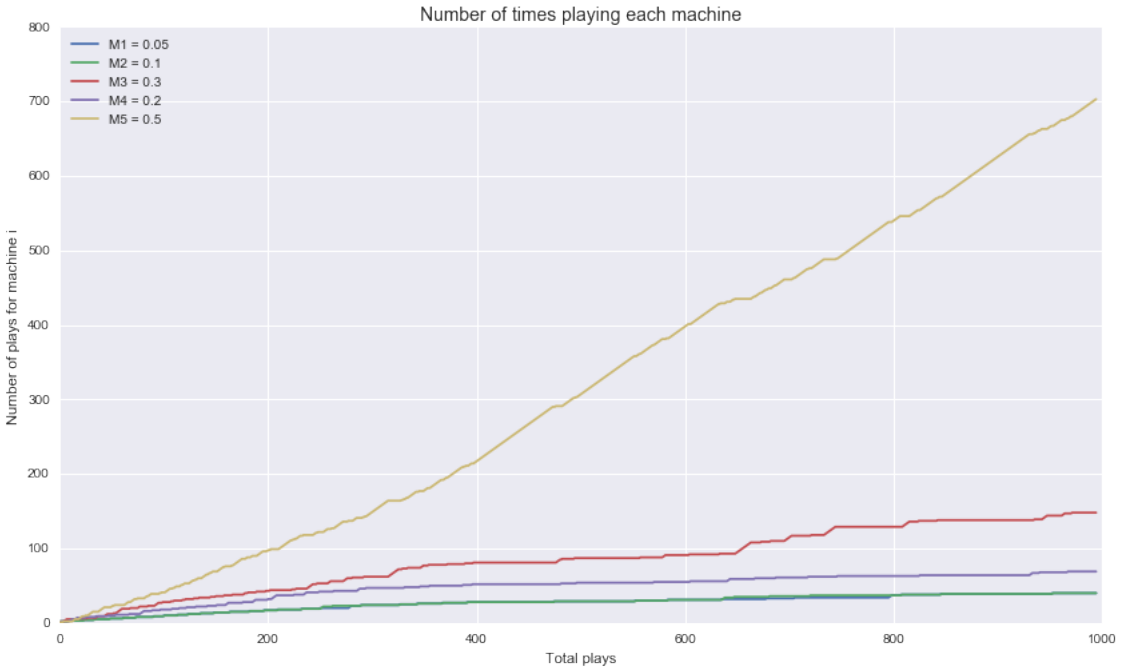

Figure 2: Switching between machines

¡¡¡¡¡¡¡**INSERT COMMENTARY**¿¿¿¿¿¿¿¿¿

3 Bayesian approaches

3.1 Bernoulli Bandit

Another popular approach to solving the MAB problem is to minimize regret using Bayesian inference. In the above sections, we assumed that rewards were distributed as a Bernoulli with some latent parameter, θ_i, specific to each machine. However we didn't make this explicit. The Bayesian approach is to put a prior distribution on θ_i and update that belief as we run the algorithm and learn which machines are better than others. Since rewards are generated from a Bernoulli distribution, a natural choice for the distribution on θ_i is a Beta. Formally:

$$\theta_i \sim \text{Beta}(\alpha_i, \beta_i)$$

$$r_i = \text{Reward from Machine i} \sim \text{Bernoulli}(\theta_i)$$

Exploiting the conjugacy of the Beta-Bernoulli model, the posterior distribution of getting a reward from machine$_i$, after playing for N iterations is:

$$\text{Beta}(\alpha_i + R_N, \beta_i + T - R_N)$$

where $R_N = \sum_{t=1}^{N} r_t$.

Before beginning to play, we assume no prior knowledge about any machine's propensity to yield a positive reward. Therefore, we initially set $\alpha_i, \beta_i = 1$, which is a common objective prior for the Beta distribution.

3.2 Thompson Sampling

The generic algorithm used for Bayesian inference is as follows:

1. For $t = 1, ..., N$
2. U(α_i, β_i) = i_t // select machine i at time t using the policy/acquisition function
3. $r_t \sim$ Bernoulli(θ_i) i // We play Machine i, and get Reward r_t
4. If $r_t = 1 : \alpha_i = \alpha_i + 1$ // if the reward was successful, increase our positive belief in machine i
5. else if $r_t = 0 : \beta_i = \beta_i + 1$ // if we didn't get a reward, increase our negative belief (i.e. failure) in machine i
6. $R_t = R_t + r_t$ // add reward at time t to running total

This algorithm provides a framework for how to think about this problem in a Bayesian way. To maximize expected rewards, we must optimally choose the policy function U.

Note that we don't really care about how accurate we become about inferences of the hidden probabilities, θ_i - for this problem we are more interested in choosing the best bandit that maximizes rewards.
Below we explore 3 ways to select the policy:

1. **Explore**: Randomly select one of the K machines, without considering how many win/losses you've seen. This is a pure eploratory strategy and is used for comparison purposes only.
2. **Exploit**: Sample each $\theta_i \sim Beta(\alpha_i, \beta_i)$. Select $\underset{i}{\text{argmax}}\ \mathbf{E}[\theta_i]$. This is called "Exploit" because we are greedily choosing the best machine at each step based on expected rewards. Note however that even if we select the best expected machine, there is a non-zero probability ($\frac{\beta_i}{\alpha_i+\beta_i}$) that it fails to generate a reward. So the amount of exploration is small but non-zero. Also note that this approach is sometimes called "Adbandit".
3. **Thompson-Sampling**: Sample each $\theta_i \sim Beta(\alpha_i, \beta_i)$. Select $\underset{i}{\text{argmax}}\ \theta_i$. This approach falls somewhere between Exploit and Explore, but is much closer to Exploit. In expectation we expect Thompson-Sampling to be very close to Exploit.

Below we present comparisons of the three approaches using the same dataset as above for UCB:

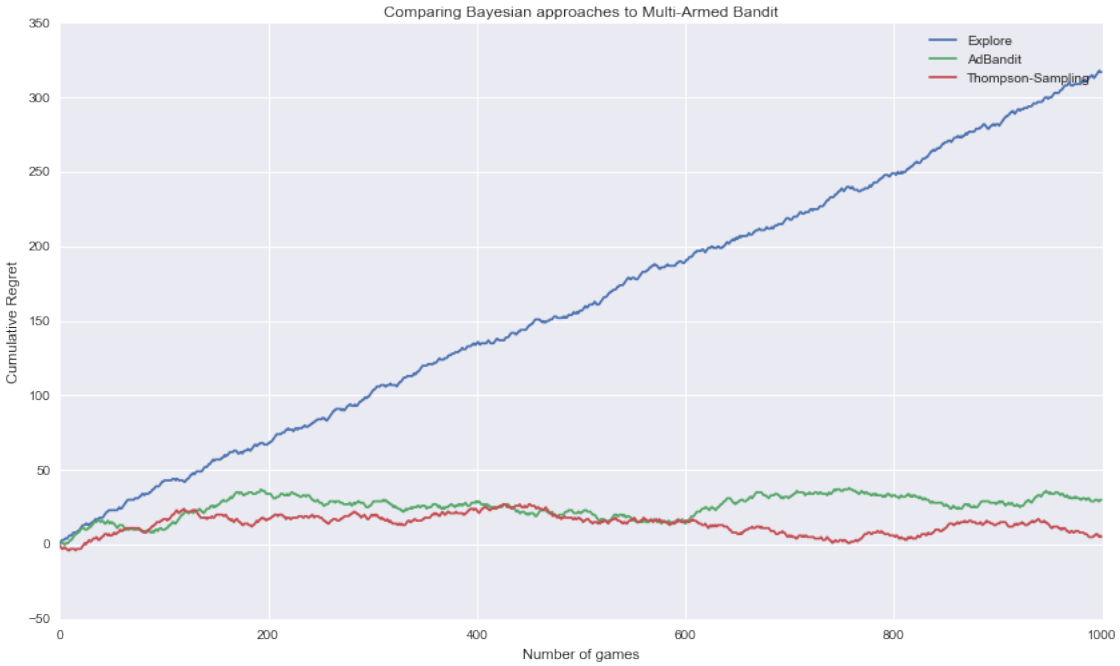

Figure 3: Cumulative Regret

3.3 Gaussian Process and Reinforcement Learning

- Lin

- assume knowledge of GPs, so no need to explain what a GP is
- quick overview of how MAB relates to reinforcement learning
- quick overview of how GPs relate to reinforcement learning
- using the mechanics of GP, what the connection between GP and MAB is
- show a chart or two like in the presentation
- this will be multiple sections

4 Empirical Comparisons

4.1 Data set

4.2 Charts

- using the same data set, compare UCB vs. epsilon vs. Thompson vs. gittins vs. GP

5 Conclusion

6 References