Chapter 9: Applications to Computing

9.8: Multi-Armed Bandits

Slides (Google Drive) Starter Code (GitHub)

Actually, for this application of bandits, we will do the problem setup before the motivation. This is because modelling problems in this bandit framework may be a bit tricky, so we'll kill two birds with one stone. We'll also see how to do "Modern Hypothesis Testing" using bandits!

9.8.1 The Multi-Armed Bandit Framework

Imagine you go to a casino in Las Vegas, and there $K = 3$ different slot machines ("Bandits" with "Arms"). (They are called bandits because they steal your money.)

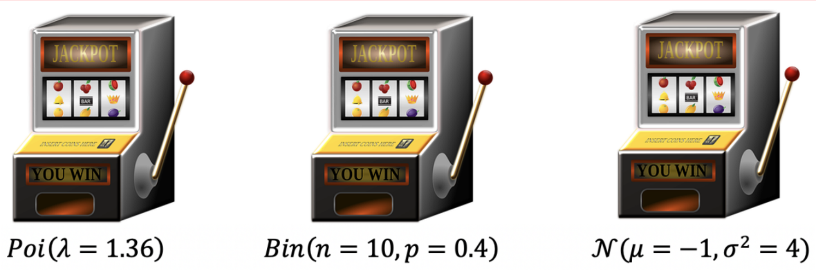
You bought some credits and can pull any slot machines, but only a total of $T = 100$ times. At each time step $t = 1, \dots, T$, you pull arm $a_t \in \{1, 2, \dots, K\}$ and observe a random reward. Your goal is to maximize your total (expected) reward after $T = 100$ pulls! The problem is: at each time step (pull), how do I decide which arm to pull based on the past history of rewards?

We make a simplifying assumption that each arm is independent of the rest, and has some reward distribution which does NOT change over time.

Here is an example you may be able to do: don't overthink it!

Example(s)

If the reward distributions are given in the image below for the $K = 3$ arms, what is the best strategy to maximize your expected reward?



Solution We can just compute the expectations of each from the distributions handout. The first machine has expectation $\lambda = 1.36$, the second has expectation $np = 4$, and the third has expectation $\mu = -1$. So to maximize our total reward, we should just always pull arm 2 because it has the best expected reward! There would be no benefit in pulling other arms at all. □

So we're done right? Well actually, we DON'T KNOW the reward distributions at all! We must **estimate** all K expectations (one per arm), WHILE simultaneously maximizing reward! This is a hard problem because we know nothing about the K reward distributions. Which arm should we pull then at each time step? Do we pull arms we know to be "good" (probably), or try other arms?

This bandit problem allows us to formally model this tradeoff between:

- **Exploitation**: Pulling arm(s) we know to be "good".
- **Exploration**: Pulling less-frequently pulled arms in the hopes they are also "good" or even better.

In this section, we will only handle the case of **Bernoulli-bandits**. That is, the reward of each arm $a \in \{1, \dots, K\}$ is $\text{Ber}(p_a)$ (i.e., we either get a reward of 1 or 0 from each machine, with possibly different probabilities). Observe that the expected reward of arm a is just p_a (expectation of Bernoulli).

The last thing we need to talk about when talking about bandits is **regret**. Regret is the difference between

- The best possible expected reward (if you always pulled the best arm).
- The actual reward you got over T arm-pulls.

Let $p^* = \arg\max_{i \in \{1,2,\dots,K\}} p_i$ denote the highest expected reward from one of the K arms. Then, the regret at time T is

$$\text{Regret}(T) = Tp^* - \text{Reward}(T)$$

where Tp^* is the reward from the best arm if you pull it T times, and $\text{Reward}(T)$ is your actual reward after T pulls. Sometimes it's easier to think about this in terms of **average regret** (divide everything by T).

$$\text{Avg-Regret}(T) = p^* - \frac{\text{Reward}(T)}{T}$$

We ideally want $\text{Avg-Regret}(T) \to 0$ as $T \to \infty$. In fact, **minimizing (average) regret is equivalent to maximizing reward** (why?). The reason we defined this is because our graphs of the plots of different algorithms we studied are best compared on such a plot with regret on the y-axis and time on the x-axis. If you look deeply at the theoretical guarantees (we won't), a lot of the times they upper-bound the regret.

The below summarizes and formalizes everything above into this so-called "Bernoulli Bandit Framework".

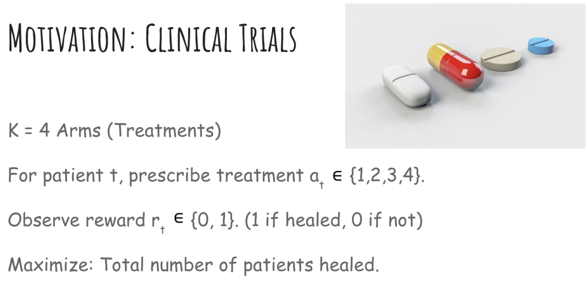
Algorithm 1 (Bernoulli) Bandit Framework

```
Have $K$ arms, where pulling arm $i \in \{1,\ldots,K\}$ gives $\text{Ber}(p_i)$ reward        ▷ $p_i$'s all unknown.
for $t = 1,\ldots,T$ do
    At time $t$, pull arm $a_t \in \{1,\ldots,K\}$.        ▷ How do we do decide which arm?
    Receive reward $r_t \sim \text{Ber}(p_{a_t})$.        ▷ Reward is either 1 or 0.
```

The focus for the rest of the entire section is: "how do we choose which arm"?

9.8.2 Motivation

Before we talk about that though, we'll discuss the motivation as promised.



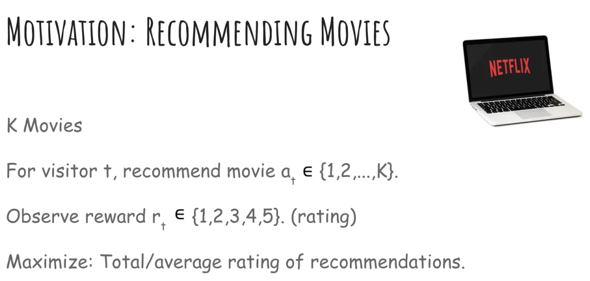
Motivation: Clinical Trials

K = 4 Arms (Treatments)

For patient t, prescribe treatment $a_t \in \{1,2,3,4\}$.

Observe reward $r_t \in \{0, 1\}$. (1 if healed, 0 if not)

Maximize: Total number of patients healed.



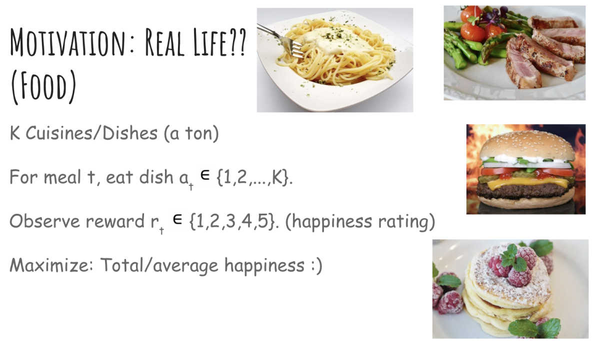
Motivation: Recommending Movies

K Movies

For visitor t, recommend movie $a_t \in \{1,2,\ldots,K\}$.

Observe reward $r_t \in \{1,2,3,4,5\}$. (rating)

Maximize: Total/average rating of recommendations.



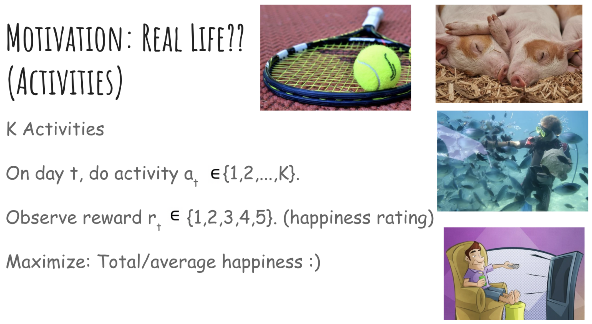
Motivation: Real Life?? (Food)

K Cuisines/Dishes (a ton)

For meal t, eat dish $a_t \in \{1,2,\ldots,K\}$.

Observe reward $r_t \in \{1,2,3,4,5\}$. (happiness rating)

Maximize: Total/average happiness :)



Motivation: Real Life?? (Activities)

K Activities

On day t, do activity $a_t \in \{1,2,\ldots,K\}$.

Observe reward $r_t \in \{1,2,3,4,5\}$. (happiness rating)

Maximize: Total/average happiness :)

As you can see above, we can model a lot of real-life problems as a bandit problem. We will learn two popular algorithms: Upper Confidence Bound (UCB) and Thompson Sampling. This is after we discuss some "intuitive" or "naive" strategies you may have yourself!

We'll actually call on a lot of our knowledge from Chapters 7 and 8! We will discuss maximum likelihood, maximum a posteriori, confidence intervals, and hypothesis testing, so you may need to brush up on those!

9.8.3 Algorithm: (Naive) Greedy Strategies

If this were a lecture, I might ask you for any ideas you may have? I encourage you to think for a minute before reading the "solution" below.

One strategy may be: pull each arm M times in the beginning, and then forever pull the best arm! This is described formally below:

Algorithm 2 Greedy (Naive) Strategy for Bernoulli Bandits

```
Choose a number of times $M$ to pull each arm initially, with $KM \leq T$.
for $i = 1, 2, \dots, K$ do
    Pull arm $i$ $M$ times, observing iid rewards $r_{i1}, \dots, r_{iM} \sim \text{Ber}(p_i)$.
    Estimate $\hat{p}_i = \frac{\sum_{j=1}^{M} r_{ij}}{M}$.        ▷ Maximum likelihood estimate!
Determine best (empirical) arm $a^* = \arg\max_{i \in \{1,2,\dots,K\}} \hat{p}_i$.        ▷ We could be wrong...
for $t = KM + 1, KM + 2, \dots, T$ do:        ▷ For the rest of time...
    Pull arm $a_t = a^*$.        ▷ Pull the same arm for the rest of time.
    Receive reward $r_t \sim \text{Ber}(p_{a_t})$.
```

Actually, this strategy is no good, because if we choose the wrong best arm, we would regret it for the rest of time! You might then say, why don't we increase M? If you do that, then you are pulling sub-optimal arms more than you should, which would not help us in maximizing reward...The problem is: we did all of our exploration FIRST, and then exploited our best arm (possibly incorrect) for the rest of time. Why don't we try to blend in exploration more? Do you have any ideas on how we might do that?

This following algorithm is called the **ε-Greedy algorithm**, because it explores with probability ε at each time step! It has the same initial setup: pull each arm M times to begin. But it does two things better than the previous algorithm:

1. It continuously updates an arm's estimated expected reward when it is pulled (even after the KM steps).

2. It explores with some probability ε (you choose). This allows you to choose in some quantitative way how to balance exploration and exploitation.

See below!

Algorithm 3 ε-Greedy Strategy for Bernoulli Bandits

```
Choose a number of times $M$ to pull each arm initially, with $KM \leq T$.
for $i = 1, 2, \dots, K$ do
    Pull arm $i$ $M$ times, observing iid rewards $r_{i1}, \dots, r_{iM} \sim \text{Ber}(p_i)$.
    Estimate $\hat{p}_i = \frac{\sum_{j=1}^{M} r_{ij}}{M}$.
for $t = KM + 1, KM + 2, \dots, T$ do:
    if $\text{Ber}(\varepsilon) == 1$: then                              ▷ With probability $\varepsilon$, explore.
        Pull arm $a_t \sim \text{Unif}(1, K)$ (discrete).               ▷ Choose a uniformly random arm.
    else                                                                 ▷ With probability $1 - \varepsilon$, exploit.
        Pull arm $a_t = \arg\max_{i \in \{1,2,\dots,K\}} \hat{p}_i$.     ▷ Choose arm with highest estimated reward.
    Receive reward $r_t \sim \text{Ber}(p_{a_t})$.
    Update $\hat{p}_{a_t}$ (using newly observed reward $r_t$).
```

However, we can do much much better! Why should we explore each arm uniformly at random, when we have a past history of rewards? Let's explore more the arms that have the *potential* to be really good! In an extreme case, if there is an arm with average reward 0.01 after 100 pulls and an arm with average reward 0.6 after only 5 pulls, should we really both explore each equally?

9.8.4 Algorithm: Upper Confidence Bound (UCB)

A great motto for this algorithm would be "optimism in the face of uncertainty". The idea of the greedy algorithm was simple: at each time step, choose the best arm (arm with highest $\hat{p}_a$). The algorithm we discuss now is very similar, but turns out to work a lot better: construct a confidence interval for $\hat{p}_a$ for each arm, and choose the one with the highest POTENTIAL to be best. That is, suppose we had three arms with the following estimates and confidence intervals at some time t:

- Arm 1: Estimate is $\hat{p}_1 = 0.75$. Confidence interval is $[0.75 - 0.10, 0.75 + 0.10] = [0.65, 0.85]$.

- Arm 2: Estimate is $\hat{p}_2 = 0.33$. Confidence interval is $[0.33 - 0.25, 0.33 + 0.25] = [0.08, 0.58]$.

- Arm 3: Estimate is $\hat{p}_3 = 0.60$. Confidence interval is $[0.60 - 0.29, 0.60 + 0.29] = [0.31, 0.89]$.

Notice all the intervals are centered at the MLE. Remember the intervals may have different widths, because the width of a confidence interval depends on how many times it has been pulled (more pulls means more confidence and hence narrower interval). Review 8.1 if you need to recall how we construct them.

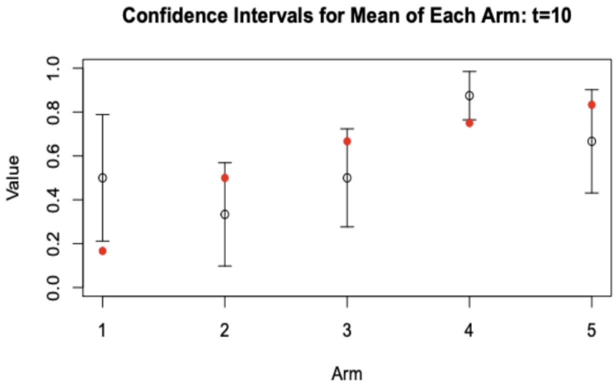
The greedy algorithm from earlier at this point in time would choose arm 1 because it has the highest estimate (0.75 is greater than 0.33 and 0.60). But our new **Upper Confidence Bound (UCB)** algorithm will choose arm 3 instead, as it has the highest possibility of being the best (0.89 is greater than 0.85 and 0.58).

Algorithm 4 UCB1 Algorithm (Upper Confidence Bound) for Bernoulli Bandits

```
for $i = 1, 2, \ldots, K$ do
    Pull arm $i$ once, observing $r_i \sim \text{Ber}(p_i)$.
    Estimate $\hat{p}_i = r_i/1$.        ▷ Each estimate $\hat{p}_i$ will *initially* either be 1 or 0.
for $t = K+1, K+2, \ldots, T$ do:
    Pull arm $a_t = \arg\max_{i \in \{1,2,\ldots,K\}} \left(\hat{p}_i + \sqrt{\frac{2\ln(t)}{N_t(i)}}\right)$, where $N_t(i)$ is the number of times arm $i$ was pulled before time $t$.
    Receive reward $r_t \sim \text{Ber}(p_{a_t})$.
    Update $N_t(a_t)$ and $\hat{p}_{a_t}$ (using newly observed reward $r_t$).
```

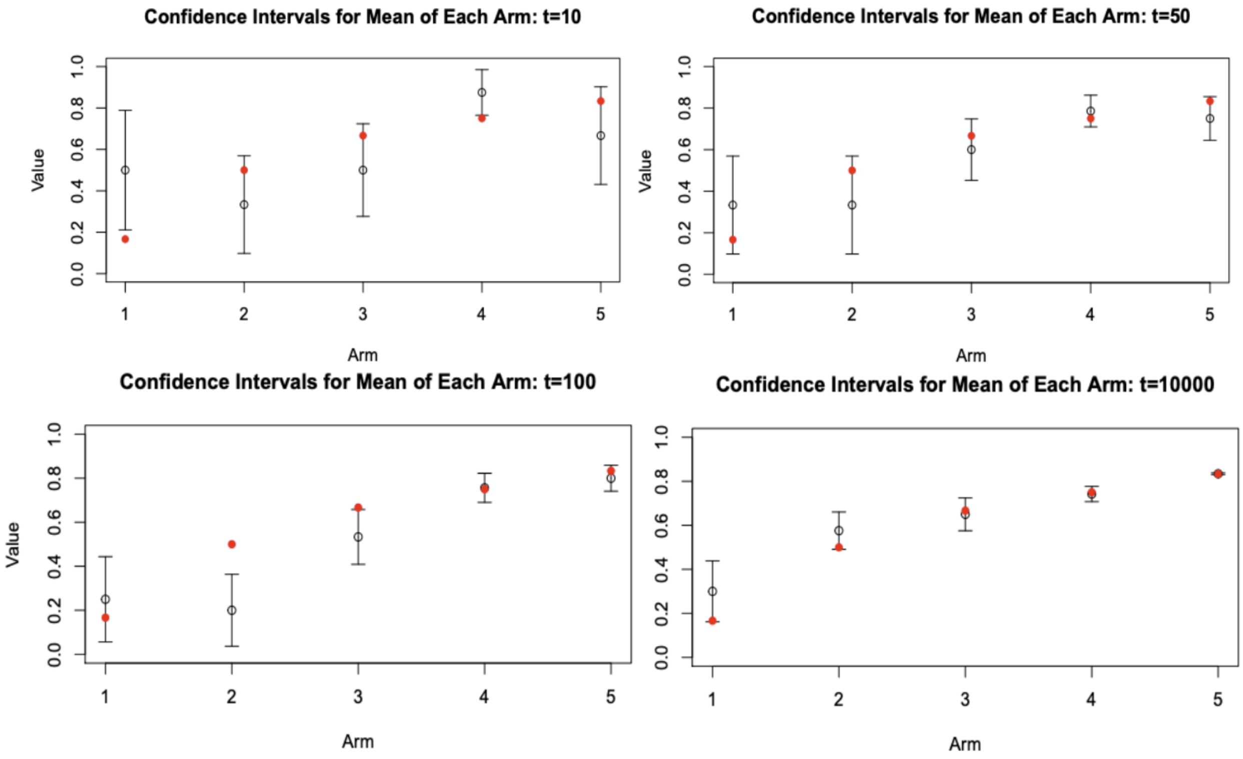
See how exploration is "baked in" now? As we pull an arm more and more, the upper confidence bound decreases. The less frequently pulled arms have a chance to have a higher UCB, despite having a lower point estimate! After the next algorithm we examine, we will visually compare and contrast the results. But before we move on, let's take a look at this visually.

Suppose we have $K = 5$ arms. The following picture depicts at time $t = 10$ what the confidence intervals may look like. The horizontal lines at the top of each arm represent the upper confidence bound, and the red dots represent the TRUE (unknown) means. The center of each confidence interval are the ESTIMATED means.



Pretty inaccurate at first right? Because it's so early on, our estimates are expected to be bad.

Now see what happens as t gets larger and larger!



Notice how the interval for the best arm (arm 5) keeps shrinking, and is the smallest one because it was pulled (exploited) so much! Clearly, arm 1 was terrible and so our estimate isn't perfect; it has the widest width since we almost never pulled it. This is the idea of UCB: basically just greedy but using upper confidence bounds!

You can go to the slides linked at the top of the section if you would like to see a step-by-step of the first few iterations of this algorithm (slides 64-86).

Note if just deleted the $+\sqrt{\frac{2\ln(t)}{N_t(i)}}$ in the 5th line of the algorithm, it would reduce to the greedy!

9.8.5 Algorithm: Thompson Sampling

This next algorithm is even better! It takes this idea of MAP (prior and posterior) into account, and ends up working extremely well. Again, we'll see how a slight change would reduce this back to the greedy algorithm.

We will assume a Beta$(1, 1)$ (uniform) prior on each unknown probability of reward. That is, we can treat our p_i's as continuous probability distributions. Remember that though with this uniform prior, the MAP and the MLE are equivalent though (pretend we saw $1 - 1 = 0$ heads and $1 - 1 = 0$ failures). However, we will not be using the posterior distribution just to get the mode, we will SAMPLE from it! Here's the idea visually.

Let's say we have $K = 3$ arms, and are at the first time step $t = 1$. We will start each arm off with a Beta$(\alpha_i = 1, \beta_i = 1)$ prior and update α_i, β_i based on the rewards we observe. We'll show the algorithm first, then use visuals to walk through it.

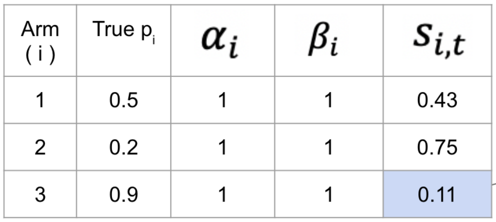
Algorithm 5 Thompson Sampling Algorithm for Beta-Bernoulli Bandits

```
For each arm $i \in \{1, \dots, K\}$, initialize $\alpha_i = \beta_i = 1$.    ▷ Set Beta($\alpha_i, \beta_i$) prior for each $p_i$.
for $t = 1, 2, \dots, T$ do:
    For each arm $i$, get sample $s_{i,t} \sim \text{Beta}(\alpha_i, \beta_i)$.    ▷ Each is a float in $[0, 1]$.
    Pull arm $a_t = \arg\max_{i \in \{1,2,\dots,K\}} s_{i,t}$.    ▷ This "bakes in" exploration!
    Receive reward $r_t \sim \text{Ber}(p_{a_t})$.
    if $r_t == 1$ then $\alpha_{a_t} \leftarrow \alpha_{a_t} + 1$.    ▷ Increment number of "successes".
    else if $r_t == 0$ then $\beta_{a_t} \leftarrow \beta_{a_t} + 1$.    ▷ Increment number of "failures".
```

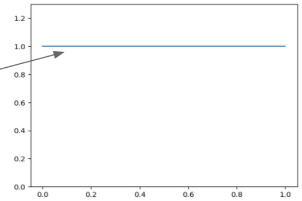
So as I mentioned earlier, each p_i is a RV which starts with a Beta$(1, 1)$ distribution. For each arm i, we keep track of α_i and β_i, where $\alpha_i - 1$ is the number of successes (number of times we got a reward of 1), and $\beta_i - 1$ is the number of failures (number of times we got a reward of 0).

For this algorithm, I would highly recommend you go to the slides linked at the top of the section if you would like to see a step-by-step of the first few iterations of this algorithm (slides 94-112). If you don't want to, we'll still walk through it below!

Let's again suppose we have $K = 3$ arms. At time $t = 1$, we sample once from each arm's Beta distribution.



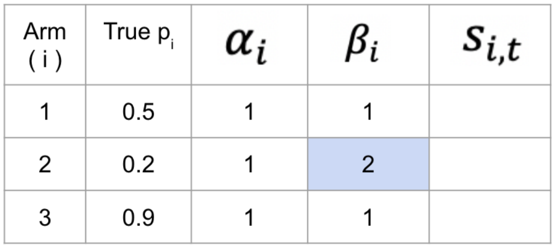
Arm (i)	True p_i	α_i	β_i	$s_{i,t}$
1	0.5	1	1	0.43
2	0.2	1	1	0.75
3	0.9	1	1	0.11



Sample from Beta(1,1) density →

We suppose the true p_i's are $0.5, 0.2$, and 0.9 for arms 1, 2, and 3 respectively (see the table). Each arm has α_i and β_i, initially 1. We get a sample from each arm's Beta distribution and just pull the arm with the largest sample! So in our first step, each has the same distribution Beta$(1, 1) = $ Unif$(0, 1)$, so each arm is equally likely to be pulled. Then, because arm 2 has the highest sample (of 0.75), we pull arm 2. The algorithm doesn't know this, but there is only a 0.2 chance of getting a 1 from arm 2 (see the table), and so let's say we happen to observe our first reward to be zero: $r_1 = 0$.

Consistent with our Beta random variable intuition and MAP, we increment our number of failures by 1 for arm 2 only.

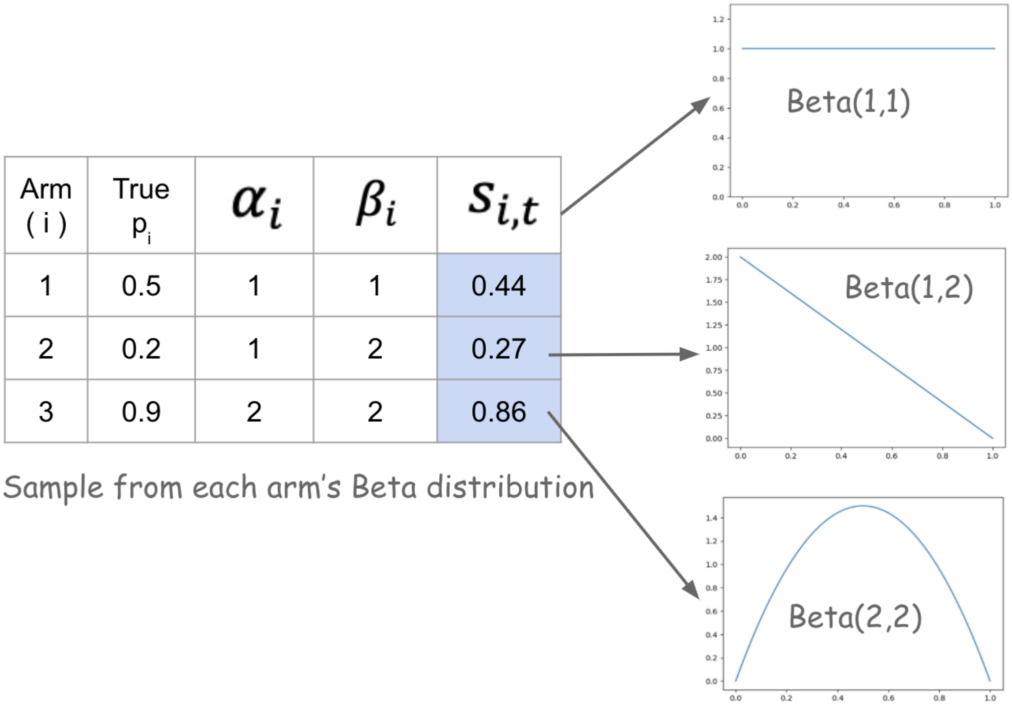


Arm (i)	True p_i	α_i	β_i	$s_{i,t}$
1	0.5	1	1	
2	0.2	1	2	
3	0.9	1	1	

Add a count of 1 to the failures :(.

At the next time step, we do the same! Sample from each arm's Beta and choose the arm with the highest sample. We'll see it for a more interesting example below after skipping a few time steps.

Now let's say we're at time step 4, and we see the following chart. Below depicts the current Beta densities for each arm, and what sample we got from each.



Arm (i)	True p_i	α_i	β_i	$s_{i,t}$
1	0.5	1	1	0.44
2	0.2	1	2	0.27
3	0.9	2	2	0.86

Sample from each arm's Beta distribution

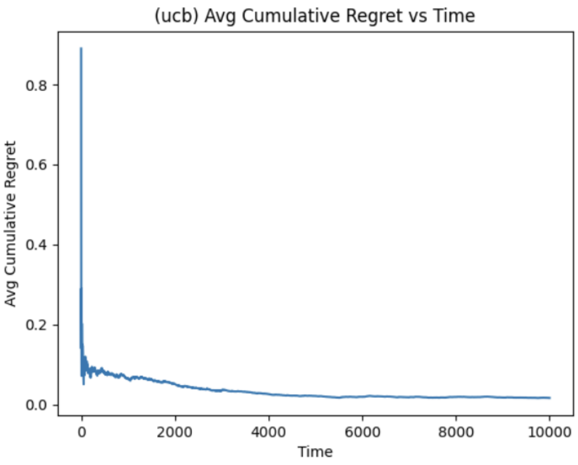
We can see from the α_i's and β_i's that we still haven't pulled arm 1 (both parameters are still at 1), we pulled arm 2 and got a reward of 0 ($\beta_2 = 2$), and we pulled arm 3 twice and got one 1 and one 0 ($\alpha_3 = \beta_3 = 2$). See the density functions below: arm 1 is equally likely to be any number in $[0, 1]$, whereas arm 2 is more likely to give a low number. Arm 3 is more certain of being in the center.

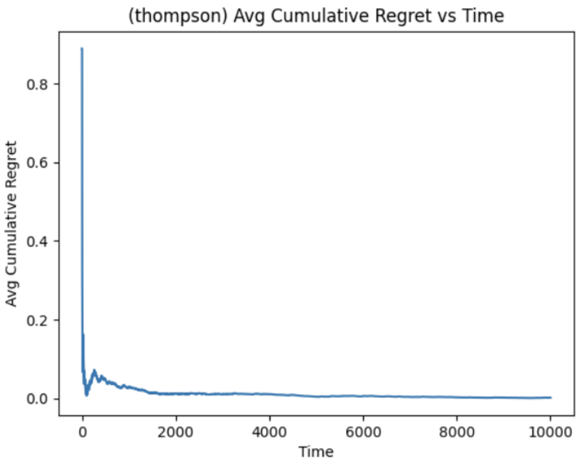
You can see that Thompson Sampling just uses this ingenious idea of **sampling** rather than just taking the MAP, and it works great! We'll see some comparisons below between UCB and Thompson sampling.

Note that with a single-line change, instead of sampling in line 3, if we just took the MAP (which equals the MLE because of our uniform prior), we would again revert back to the greedy algorithm! The exploration comes from the sampling, which works out great for us!

9.8.6 Comparison of Methods

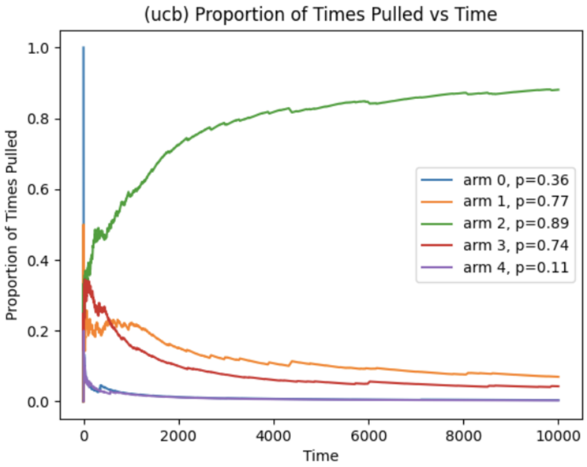
See the UCB and Thompson Sampling's average regret over time:

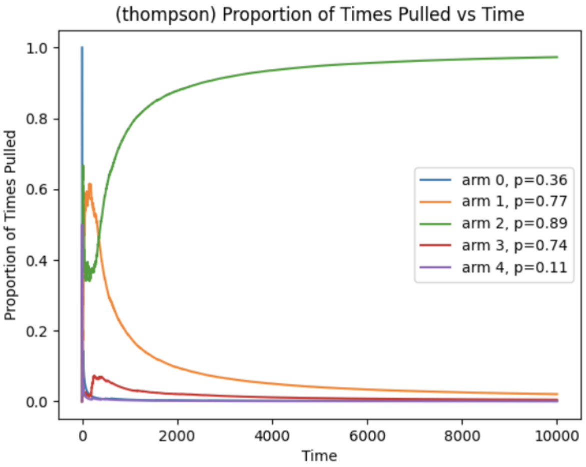




It might be a bit hard to see, but notice Thompson sampling's regret got close to 0 a lot faster than UCB. UCB happened around time 5000, and TS happened around time 2000. The reason why Thompson sampling might be "better" is unfortunately out of scope.

Below is my favorite visualization of all. On the x-axis we have time, and on the y-axis, we have the proportion of time each arm was pulled (there were $K = 5$ arms). Notice how arm 2 (green) has the highest true expected reward at 0.89, and how quickly Thompson sampling discovered it and starting exploiting it.





9.8.7 Modern Hypothesis Testing

Not only do bandits solve all the applications we talked about earlier, it actually provides a modernized way to conduct hypothesis tests.

Let's say a large tech company wants to experiment releasing a new feature/modification.

They assign

- 99% of population to control group (current feature)
- 1% to experimental group (new feature).

This has the following consequences:

- If the new feature is "bad", very small percentage of the population sees it, so company protects itself.
- If the new feature is "good", very small percentage of the population sees it, so company may lose revenue.

We would perform a two-sample hypothesis test (called an "A/B Test" in industry) now to compare the means of some metric we cared about (click-through rate for example), and determine whether we could reject the null hypothesis and statistically prove that the new feature performs better. Can we do better though? Can we adaptively assign subjects to each group based on how each is performing rather than deciding at the beginning? Yes, let's use bandits!

Let's use the Bernoulli-bandit framework with just $K = 2$ arms:

- Arm 1: Current Feature
- Arm 2: New feature

When feature is requested by some user, use Multi-Armed Bandit algorithm to decide which feature to show! We can have any number of arms.

Here are the benefits and drawbacks of using Traditional A/B Testing vs Multi-Armed bandits. Each has their own advantages, and you should carefully consider which approach to take before arbitrarily deciding!

When to use Traditional A/B Testing:

- Need to collect data for critical business decisions.
- Need statistical confidence in all your results and impact. Want to learn even about treatments that didn't perform well.
- The reward is not immediate (e.g., if drug testing, don't have time to wait for each patient to finish before experimenting with next patient).
- Optimize/measure multiple metrics, not just one.

When to use Multi-Armed Bandits:

1. No need for interpreting results, just maximize reward (typically revenue/engagement)
2. The opportunity cost is high (if advertising a car, losing a conversion is $\geq$\$20,000)
3. Can add/remove arms in the middle of an experiment! Cannot do with A/B tests.

The study of Multi-Armed Bandits can be categorized as:

- Statistics
- Optimization
- "Reinforcement Learning" (subfield of Machine Learning)