Math ∩ Programming

Optimism in the Face of Uncertainty: the UCB1 Algorithm

Posted on October 28, 2013 by j2kun

(https://jeremykun.files.wordpress.com/2013/10/startups.jpg)

The software world is always atwitter with predictions on the next big piece of technology. And a lot of chatter (https://www.hnsearch.com/search#request/submissions&q=YC&start=0) focuses on what venture capitalists express interest in. As an investor, how do you pick a good company to invest in? Do you notice quirky names like "Kaggle" and "Meebo," require deep technical abilities, or value a charismatic sales pitch?

This author personally believes we're not thinking as big as we should be (http://vimeo.com/71278954) when it comes to innovation in software engineering and computer science, and that as a society we should value big pushes forward much more than we do. But making safe investments is almost always at odds with innovation. And so every venture capitalist faces the following question. When do you focus investment in those companies that have proven to succeed, and when do you explore new options for growth? A successful venture capitalist must strike a fine balance between this kind of exploration and exploitation. Explore too much and you won't make enough profit to sustain yourself. Narrow your view too much and you will miss out on opportunities whose return surpasses any of your current prospects.

In life and in business there is no correct answer on what to do, partly because we just don't have a good understanding of how the world works (or markets, or people, or the weather (http://xkcd.com/1126/)). In mathematics, however, we can meticulously craft settings that have solid answers. In this post we'll describe one such scenario, the so-called *multi-armed bandit* problem, and a simple algorithm called UCB1 which performs close to optimally. Then, in a future post, we'll analyze the algorithm on some real world data.

As usual, all of the code (https://github.com/j2kun/ucb1) used in the making of this post are available for download on this blog's Github page (https://github.com/j2kun/).

Multi-Armed Bandits

The multi-armed bandit scenario is simple to describe, and it boils the exploration-exploitation tradeoff down to its purest form.

Suppose you have a set of K actions labeled by the integers $\{1, 2, \dots, K\}$. We call these *actions* in the abstract, but in our minds they're slot machines. We can then play a game where, in each round, we choose an action (a slot machine to play), and we observe the resulting payout. Over many rounds, we might explore the machines by trying some at random. Assuming the machines are not identical, we naturally play machines that seem to pay off well more frequently to try to maximize our total winnings.


(https://jeremykun.files.wordpress.com/2013/10/slot-machines.jpg)
Exploit away, you lucky ladies.

This is the most general description of the game we could possibly give, and every bandit learning problem has these two components: actions and rewards. But in order to get to a concrete problem that we can reason about, we need to specify more details. Bandit learning is a large tree of variations and this is the point at which the field ramifies. We presently care about two of the main branches.

How are the rewards produced? There are many ways that the rewards could work. One nice option is to have the rewards for action i be drawn from a fixed distribution D_i (a different reward distribution for each action), and have the draws be independent across rounds and across actions. This is called the *stochastic* setting and it's what we'll use in this post. Just to pique the reader's interest, here's the alternative: instead of having the rewards be chosen randomly, have them be *adversarial*. That is, imagine a casino owner *knows your algorithm* and your internal beliefs about which machines are best at any given

time. He then fixes the payoffs of the slot machines in advance of each round to screw you up! This sounds dismal, because the casino owner could just make all the machines pay nothing every round. But actually we can design good algorithms for this case, but "good" will mean something different than absolute winnings. And so we must ask:

How do we measure success? In both the stochastic and the adversarial setting, we're going to have a hard time coming up with any theorems about the performance of an algorithm if we care about how much absolute reward is produced. There's nothing to stop the distributions from having terrible expected payouts, and nothing to stop the casino owner from intentionally giving us no payout. Indeed, the problem lies in our measurement of success. A better measurement, which we can apply to both the stochastic and adversarial settings, is the notion of *regret*. We'll give the definition for the stochastic case, and investigate the adversarial case in a future post.

Definition: Given a player algorithm A and a set of actions $\{1, 2, \dots, K\}$, the *cumulative regret* of A in rounds $1, \dots, T$ is the difference between the expected reward of the best action (the action with the highest expected payout) and the expected reward of A for the first T rounds.

We'll add some more notation shortly to rephrase this definition in symbols, but the idea is clear: we're competing against the best action. Had we known it ahead of time, we would have just played it every single round. Our notion of success is not in how well we do absolutely, but in how well we do *relative to what is feasible*.

Notation

Let's go ahead and draw up some notation. As before the actions are labeled by integers $\{1, \dots, K\}$. The *reward* of action i is a $[0, 1]$-valued random variable X_i distributed according to an unknown distribution and possessing an unknown expected value μ_i. The game progresses in rounds $t = 1, 2, \dots$ so that in each round we have different random variables $X_{i,t}$ for the reward of action i in round t (in particular, $X_{i,t}$ and $X_{i,s}$ are identically distributed). The $X_{i,t}$ are independent as both t and i vary, although when i varies the distribution changes.

So if we were to play action 2 over and over for T rounds, then the total payoff would be the random variable $G_2(T) = \sum_{t=1}^{T} X_{2,t}$. But by independence across rounds and the linearity of expectation, the expected payoff is just $\mu_2 T$. So we can describe the *best action* as the action with the highest expected payoff. Define

$$\mu^* = \max_{1 \leq i \leq K} \mu_i$$

We call the action which achieves the maximum i^*.

A *policy* is a randomized algorithm A which picks an action in each round based on the history of chosen actions and observed rewards so far. Define I_t to be the action played by A in round t and $P_i(n)$ to be the number of times we've played action i in rounds $1 \leq t \leq n$. These are both random variables. Then the cumulative payoff for the algorithm A over the first T rounds, denoted $G_A(T)$, is just

$$G_A(T) = \sum_{t=1}^{T} X_{I_t,t}$$

and its expected value is simply

$$\mathbb{E}(G_A(T)) = \mu_1 \mathbb{E}(P_1(T)) + \cdots + \mu_K \mathbb{E}(P_K(T)).$$

Here the expectation is taken over all random choices made by the policy and over the distributions of rewards, and indeed both of these can affect how many times a machine is played.

Now the *cumulative regret* of a policy A after the first T steps, denoted $R_A(T)$ can be written as

$$R_A(T) = G_{i^*}(T) - G_A(T)$$

And the goal of the policy designer for this bandit problem is to minimize the expected cumulative regret, which by linearity of expectation is

$$\mathbb{E}(R_A(T)) = \mu^* T - \mathbb{E}(G_A(T)).$$

Before we continue, we should note that there are theorems concerning *lower bounds* for expected cumulative regret. Specifically, for this problem it is known that no algorithm can guarantee an expected cumulative regret better than $\Omega(\sqrt{KT})$. It is also known that there are algorithms that guarantee no worse than $O(\sqrt{KT})$ expected regret. The algorithm we'll see in the next section, however, only guarantees $O(\sqrt{KT \log T})$. We present it on this blog because of its simplicity and ubiquity in the field.

The UCB1 Algorithm

The policy we examine is called UCB1, and it can be summed up by the principle of **optimism in the face of uncertainty**. That is, despite our lack of knowledge in what actions are best we will construct an optimistic guess as to how good the expected payoff of each action is, and pick the action with the highest guess. If our guess is wrong, then our optimistic guess will quickly decrease and we'll be compelled to switch to a different action. But if we pick well, we'll be able to exploit that action and incur little regret. In this way we balance exploration and exploitation.

The formalism is a bit more detailed than this, because we'll need to ensure that we don't rule out good actions that fare poorly early on. Our "optimism" comes in the form of an *upper confidence bound* (hence the acronym UCB). Specifically, we want to know with high probability that the true expected payoff of an action μ_i is less than our prescribed upper bound. One general (distribution independent) way to do that is to use the Chernoff-Hoeffding inequality (https://jeremykun.com/2013/04/15/probabilistic-bounds-a-primer/).

As a reminder, suppose $Y_1, \ldots, Y_n$ are independent random variables whose values lie in $[0, 1]$ and whose expected values are μ_i. Call $Y = \frac{1}{n}\sum_i Y_i$ and $\mu = \mathbb{E}(Y) = \frac{1}{n}\sum_i \mu_i$. Then the Chernoff-Hoeffding inequality gives an exponential upper bound on the probability that the value of Y deviates from its mean. Specifically,

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

For us, the Y_i will be the payoff variables for a single action j in the rounds for which we choose action j. Then the variable Y is just the empirical average payoff for action j over all the times we've tried it. Moreover, a is our one-sided upper bound (and as a lower bound, sometimes). We can then solve this equation for a to find an upper bound big enough to be confident that we're within a of the true mean.

Indeed, if we call n_j the number of times we played action j thus far, then $n = n_j$ in the equation above, and using $a = a(j, T) = \sqrt{2\log(T)/n_j}$ we get that $\mathrm{P}(Y > \mu + a) \leq T^{-4}$, which converges to zero very quickly as the number of rounds played grows. We'll see this pop up again in the algorithm's analysis below. But before that note two things. First, assuming we don't play an action j, its upper bound a grows in the number of rounds. This means that we never permanently rule out an action no matter how poorly it performs. If we get extremely unlucky with the optimal action, we will eventually be convinced to try it again. Second, the probability that our upper bound is wrong decreases in the number of rounds independently of how many times we've played the action. That is because our upper bound $a(j, T)$ is getting bigger for actions we haven't played; any round in which we play an action j, it must be that $a(j, T + 1) = a(j, T)$, although the empirical mean will likely change.

With these two facts in mind, we can formally state the algorithm and intuitively understand why it should work.

UCB1:
Play each of the K actions once, giving initial values for empirical mean payoffs $\overline{x}_i$ of each action i.
For each round $t = K, K + 1, \ldots$:

Let n_j represent the number of times action j was played so far.
Play the action j maximizing $\overline{x}_j + \sqrt{2\log t/n_j}$.
Observe the reward $X_{j,t}$ and update the empirical mean for the chosen action.

And that's it. Note that we're being super stateful here: the empirical means x_j change over time, and we'll leave this update implicit throughout the rest of our discussion (sorry, functional programmers, but the notation is horrendous otherwise).

Before we implement and test this algorithm, let's go ahead and prove that it achieves nearly optimal regret. The reader uninterested in mathematical details should skip the proof, but the discussion of the theorem itself is important. If one wants to use this algorithm in real life, one needs to understand the guarantees it provides in order to adequately quantify the risk involved in using it.

Theorem: Suppose that UCB1 is run on the bandit game with K actions, each of whose reward distribution $X_{i,t}$ has values in [0,1]. Then its expected cumulative regret after T rounds is at most $O(\sqrt{KT\log T})$.

Actually, we'll prove a more specific theorem. Let Δ_i be the difference $\mu^* - \mu_i$, where μ^* is the expected payoff of the best action, and let Δ be the minimal nonzero Δ_i. That is, Δ_i represents how suboptimal an action is and Δ is the suboptimality of the second best action. These constants are called *problem-dependent constants.* The theorem we'll actually prove is:

Theorem: Suppose UCB1 is run as above. Then its expected cumulative regret $\mathbb{E}(R_{\mathrm{UCB1}}(T))$ is at most

$$8 \sum_{i:\mu_i<\mu^*} \frac{\log T}{\Delta_i} + \left(1 + \frac{\pi^2}{3}\right) \left(\sum_{j=1}^{K} \Delta_j\right)$$

Okay, this looks like one nasty puppy, but it's actually not that bad. The first term of the sum signifies that we expect to play any suboptimal machine about a logarithmic number of times, roughly scaled by how hard it is to distinguish from the optimal machine. That is, if Δ_i is small we will require more tries to know that action i is suboptimal, and hence we will incur more regret. The second term represents a small constant number (the $1 + \pi^2/3$ part) that caps the number of times we'll play suboptimal machines in excess of the first term due to unlikely events occurring. So the first term is like our expected losses, and the second is our risk.

But note that this is a *worst-case bound* on the regret. We're not saying we will achieve this much regret, or anywhere near it, but that UCB1 simply cannot do worse than this. Our hope is that in practice UCB1 performs much better.

Before we prove the theorem, let's see how derive the $O(\sqrt{KT \log T})$ bound mentioned above. This will require familiarity with multivariable calculus, but such things must be endured like ripping off a band-aid. First consider the regret as a function $R(\Delta_1, \ldots, \Delta_K)$ (excluding of course Δ^*), and let's look at the worst case bound by maximizing it. In particular, we're just finding the problem with the parameters which screw our bound as badly as possible, The gradient of the regret function is given by

$$\frac{\partial R}{\partial \Delta_i} = -\frac{8 \log T}{\Delta_i^2} + 1 + \frac{\pi^2}{3}$$

and it's zero if and only if for each i, $\Delta_i = \sqrt{\frac{8 \log T}{1+\pi^2/3}} = O(\sqrt{\log T})$. However this is a *minimum* of the regret bound (the Hessian is diagonal and all its eigenvalues are positive). Plugging in the $\Delta_i = O(\sqrt{\log T})$ (which are all the same) gives a total bound of $O(K\sqrt{\log T})$. If we look at the only possible endpoint (the $\Delta_i = 1$), then we get a local maximum of $O(K\sqrt{\log T})$. But this isn't the $O(\sqrt{KT \log T})$ we promised, what gives? Well, this upper bound grows arbitrarily large as the Δ_i go to zero. But at the same time, if all the Δ_i are small, then we shouldn't be incurring much regret because we'll be picking actions that are close to optimal!

Indeed, if we assume for simplicity that all the $\Delta_i = \Delta$ are the same, then another trivial regret bound is ΔT (why?). The true regret is hence the minimum of this regret bound and the UCB1 regret bound: as the UCB1 bound degrades we will eventually switch to the simpler bound. That will be a non-differentiable switch (and hence a critical point) and it occurs at $\Delta = O(\sqrt{(K \log T)/T})$. Hence the regret bound at the switch is $\Delta T = O(\sqrt{KT \log T})$, as desired.

Proving the Worst-Case Regret Bound

Proof. The proof works by finding a bound on $P_i(T)$, the expected number of times UCB chooses an action up to round T. Using the Δ notation, the regret is then just $\sum_i \Delta_i \mathbb{E}(P_i(T))$, and bounding the P_i's will bound the regret.

Recall the notation for our upper bound $a(j,T) = \sqrt{2\log T/P_j(T)}$ and let's loosen it a bit to $a(y,T) = \sqrt{2\log T/y}$ so that we're allowed to "pretend" a action has been played y times. Recall further that the random variable I_t has as its value the index of the machine chosen. We denote by $\chi(E)$ the indicator random variable for the event E. And remember that we use an asterisk to denote a quantity associated with the optimal action (e.g., $\overline{x}^*$ is the empirical mean of the optimal action).

Indeed for any action i, the only way we know how to write down $P_i(T)$ is as

$$P_i(T) = 1 + \sum_{t=K}^{T} \chi(I_t = i)$$

The 1 is from the initialization where we play each action once, and the sum is the trivial thing where just count the number of rounds in which we pick action i. Now we're just going to pull some number $m-1$ of plays out of that summation, keep it variable, and try to optimize over it. Since we might play the action fewer than m times overall, this requires an inequality.

$$P_i(T) \leq m + \sum_{t=K}^{T} \chi(I_t = i \text{ and } P_i(t-1) \geq m)$$

These indicator functions should be read as sentences: we're just saying that we're picking action i in round t and we've already played i at least m times. Now we're going to focus on the inside of the summation, and come up with an event that happens at least as frequently as this one to get an upper bound. Specifically, saying that we've picked action i in round t means that the upper bound for action i exceeds the upper bound for every other action. In particular, this means its upper bound exceeds the upper bound of the best action (and i might coincide with the best action, but that's fine). In notation this event is

$$\overline{x}_i + a(P_i(t), t-1) \geq \overline{x}^* + a(P^*(T), t-1)$$

Denote the upper bound $\overline{x}_i + a(i,t)$ for action i in round t by $U_i(t)$. Since this event must occur every time we pick action i (though not necessarily vice versa), we have

$$P_i(T) \leq m + \sum_{t=K}^{T} \chi(U_i(t-1) \geq U^*(t-1) \text{ and } P_i(t-1) \geq m)$$

We'll do this process again but with a slightly more complicated event. If the upper bound of action i exceeds that of the optimal machine, it is also the case that the maximum upper bound for action i we've seen after the first m trials exceeds the minimum upper bound we've seen on the optimal machine (ever). But on round t we don't know how many times we've played the optimal machine, nor do we even know how many times we've played machine i (except that it's more than m). So we try all possibilities and look at minima and maxima. This is a pretty crude approximation, but it will allow us to write things in a nicer form.

Denote by $\overline{x}_{i,s}$ the random variable for the empirical mean after playing action i a total of s times, and $\overline{x}^*_s$ the corresponding quantity for the optimal machine. Realizing everything in notation, the above argument proves that

$$P_i(T) \leq m + \sum_{t=K}^{T} \chi\left(\max_{m \leq s < t} \overline{x}_{i,s} + a(s, t-1) \geq \min_{0 < s' < t} \overline{x}^*_{s'} + a(s', t-1)\right)$$

Indeed, at each t for which the max is greater than the min, there will be at least one pair s, s' for which the values of the quantities inside the max/min will satisfy the inequality. And so, even worse, we can just count the number of pairs s, s' for which it happens. That is, we can expand the event above into the double sum which is at least as large:

$$P_i(T) \leq m + \sum_{t=K}^{T} \sum_{s=m}^{t-1} \sum_{s'=1}^{t-1} \chi\left(\overline{x}_{i,s} + a(s, t-1) \geq \overline{x}^*_{s'} + a(s', t-1)\right)$$

We can make one other odd inequality by increasing the sum to go from $t = 1$ to ∞. This will become clear later, but it means we can replace $t - 1$ with t and thus have

$$P_i(T) \leq m + \sum_{t=1}^{\infty} \sum_{s=m}^{t-1} \sum_{s'=1}^{t-1} \chi\left(\overline{x}_{i,s} + a(s, t) \geq \overline{x}^*_{s'} + a(s', t)\right)$$

Now that we've slogged through this mess of inequalities, we can actually get to the heart of the argument. Suppose that this event actually happens, that $\overline{x}_{i,s} + a(s,t) \geq \overline{x}^*_{s'} + a(s',t)$. Then what can we say? Well, consider the following three events:

(1) $\overline{x}^*_{s'} \leq \mu^* - a(s',t)$
(2) $\overline{x}_{i,s} \geq \mu_i + a(s,t)$
(3) $\mu^* < \mu_i + 2a(s,t)$

In words, (1) is the event that the empirical mean of the optimal action is less than the lower confidence bound. By our Chernoff bound argument earlier, this happens with probability t^{-4}. Likewise, (2) is the event that the empirical mean payoff of action i is larger than the upper confidence bound, which also occurs with probability t^{-4}. We will see momentarily that (3) is impossible for a well-chosen m (which is why we left it variable), but in any case the claim is that one of these three events must occur. For if they are all false, we have

$$\underbrace{\overline{x}_{i,s} + a(s,t) \geq \overline{x}^*_{s'} + a(s',t)}_{\text{assumed}} > \underbrace{\mu^* - a(s',t) + a(s',t) = \mu^*}_{(1) \text{ is false}}$$

and

$$\underbrace{\mu_i + 2a(s,t)}_{(2) \text{ is false}} > \underbrace{\overline{x}_{i,s} + a(s,t) \geq \overline{x}^*_{s'} + a(s',t)}_{\text{assumed}}$$

But putting these two inequalities together gives us precisely that (3) is true:

$$\mu^* < \mu_i + 2a(s,t)$$

This proves the claim.

By the union bound (http://en.wikipedia.org/wiki/Boole's_inequality), the probability that at least one of these events happens is $2t^{-4}$ plus whatever the probability of (3) being true is. But as we said, we'll pick m to make (3) always false. Indeed m depends on which action i is being played, and if $s \geq m > 8 \log T / \Delta_i^2$ then $2a(s,t) \leq \Delta_i$, and by the definition of Δ_i we have

$$\mu^* - \mu_i - 2a(s,t) \geq \mu^* - \mu_i - \Delta_i = 0.$$

Now we can finally piece everything together. The expected value of an event is just its probability of occurring, and so

$$\begin{aligned}
\mathbb{E}(P_i(T)) &\leq m + \sum_{t=1}^{\infty}\sum_{s=m}^{t}\sum_{s'=1}^{t} \mathrm{P}(\overline{x}_{i,s} + a(s,t) \geq \overline{x}^*_{s'} + a(s',t)) \\
&\leq \left\lceil \frac{8\log T}{\Delta_i^2} \right\rceil + \sum_{t=1}^{\infty}\sum_{s=m}^{t}\sum_{s'=1}^{t} 2t^{-4} \\
&\leq \frac{8\log T}{\Delta_i^2} + 1 + \sum_{t=1}^{\infty}\sum_{s=1}^{t}\sum_{s'=1}^{t} 2t^{-4} \\
&= \frac{8\log T}{\Delta_i^2} + 1 + 2\sum_{t=1}^{\infty} t^{-2} \\
&= \frac{8\log T}{\Delta_i^2} + 1 + \frac{\pi^2}{3}
\end{aligned}$$

The second line is the Chernoff bound we argued above, the third and fourth lines are relatively obvious algebraic manipulations, and the last equality uses the classic solution to Basel's problem (http://en.wikipedia.org/wiki/Basel_problem). Plugging this upper bound in to the regret formula we gave in the first paragraph of the proof establishes the bound and proves the theorem.

□

Implementation and an Experiment

The algorithm is about as simple to write in code as it is in pseudocode. The confidence bound is trivial to implement (though note we index from zero):

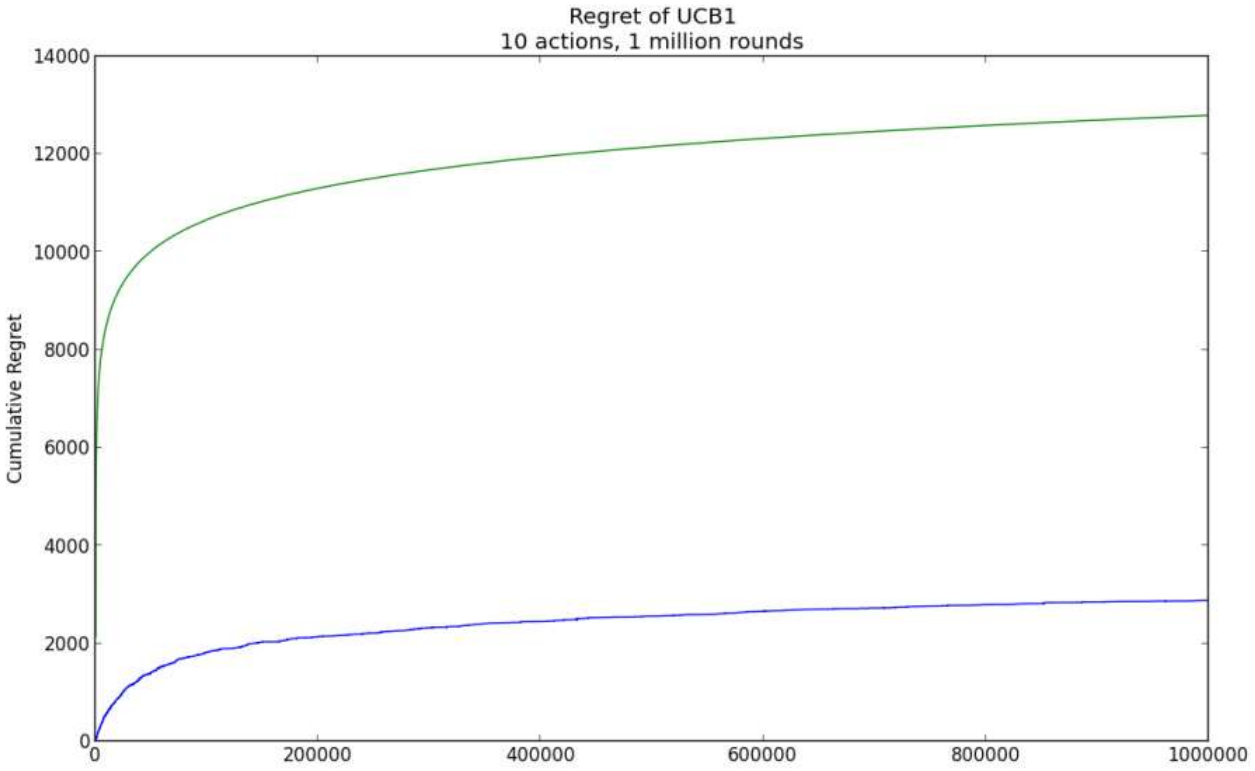
```
1  def upperBound(step, numPlays):
2      return math.sqrt(2 * math.log(step + 1) / numPlays)
```

And the full algorithm is quite short as well. We define a function ub1, which accepts as input the number of actions and a function reward which accepts as input the index of the action and the time step, and draws from the appropriate reward distribution. Then implementing ub1 is simply a matter of keeping track of empirical averages and an argmax. We implement the function as a Python generator, so one can observe the steps of the algorithm and keep track of the confidence bounds and the cumulative regret.

```
def ucb1(numActions, reward):
   payoffSums = [0] * numActions
   numPlays = [1] * numActions
   ucbs = [0] * numActions

   # initialize empirical sums
   for t in range(numActions):
      payoffSums[t] = reward(t,t)
      yield t, payoffSums[t], ucbs

   t = numActions

   while True:
      ucbs = [payoffSums[i] / numPlays[i] + upperBound(t, numPlays[i]) for i in range(numActions)]
      action = max(range(numActions), key=lambda i: ucbs[i])
      theReward = reward(action, t)
      numPlays[action] += 1
      payoffSums[action] += theReward

      yield action, theReward, ucbs
      t = t + 1
```

The heart of the algorithm is the second part, where we compute the upper confidence bounds and pick the action maximizing its bound.

We tested this algorithm on synthetic data. There were ten actions and a million rounds, and the reward distributions for each action were uniform from $[0, 1]$, biased by $1/k$ for some $5 \leq k \leq 15$. The regret and theoretical regret bound are given in the graph below.



(https://jeremykun.files.wordpress.com/2013/10/ucb1-simple-example.png)
The regret of ucb1 run on a simple example. The blue curve is the cumulative regret of the algorithm after a given number of steps. The green curve is the theoretical upper bound on the regret.

Note that both curves are logarithmic, and that the actual regret is quite a lot smaller than the theoretical regret. The code used to produce the example (https://github.com/j2kun/ucb1) and image are available on this blog's Github page (https://github.com/j2kun/).

Next Time

One interesting assumption that UCB1 makes in order to do its magic is that the payoffs are stochastic and independent across rounds. Next time (https://jeremykun.com/2013/11/08/adversarial-bandits-and-the-exp3-algorithm/) we'll look at an algorithm that assumes the payoffs are instead *adversarial*, as we described earlier. Surprisingly, in the adversarial case we can do about as well as the stochastic case. Then, we'll experiment with the two algorithms on a real-world application (https://jeremykun.com/2013/12/09/bandits-and-stocks/).

Until then!

This entry was posted in Algorithms, Learning Theory, Probability Theory, Statistics and tagged bandit learning, big-o notation, calculus, confidence bounds, machine learning, mathematics, programming, python, random variables, randomized algorithm. Bookmark the permalink.

36 thoughts on "Optimism in the Face of Uncertainty: the UCB1 Algorithm"

1. **Himadri**

   December 13, 2013 at 10:49 am • Reply

   Lucid explanation. Thanks

2. **Seeker**

   May 8, 2014 at 5:18 pm • Reply

   Hi Jeremy,

   Regarding $X_{t,i}$ I should add since you assume they have the same mean it appears to me that you are assuming [implicitly] that they are from the same distribution which from my point of view justifies replaceing it with X_i. Am I wrong?

   - **j2kun**

     May 8, 2014 at 5:45 pm • Reply

     I think it's notationally more appropriate to have separate random variables, each drawn from the same distribution, because there is a difference between and as random variables. For example, one is affected by statements like the Central Limit Theorem.

     - **Seeker**

       May 8, 2014 at 6:15 pm

I agree with you that these two are different but even in that case what you need is to declare that there are stationary process; What seemed confusing for me was the fact that you assumed the “independently” but not “identically” of i.i.d. random variables above, right?

Btw, thanks for these great articles.

- **j2kun**

  May 8, 2014 at 6:36 pm

  I hope the identical part was clear from the prose.

3. **Seeker**

   May 8, 2014 at 6:14 pm • Reply

   I agree with you that these two are different but even in that case what you need is to declare that there are stationary process; What seemed confusing for me was the fact that you assumed the “independently” but not “identically” of i.i.d. random variables above, right?

   Btw, thanks for these great articles.

4. **Seeker**

   May 8, 2014 at 6:54 pm • Reply

   I see. I am just so skeptic 🙂 . At least now I am sure the claim was exactly about i.i.d.s

5. **shubin**

   August 6, 2014 at 1:40 pm • Reply

   i think you have a code error in your python code. in ucb1 function, you never actually applied part 2 algorithms because it stops at the initialize empirical sums step.

   - **j2kun**

     August 6, 2014 at 7:40 pm • Reply

     The python function is a generator, so you have to iterate through the generator in order for it to run all the steps. For example, you might run:

     for roundinfo in ucb1(…):
     print(roundinfo)

6. **Jesse**

   October 10, 2014 at 10:23 am • Reply

   This is a really well-written explanation of UCB1, Jeremy. Thanks!

   Just had one question though:

   How do you know when to stop the experiment? Is the computed upper bound value (Math.sqrt(2*logT/n_j)) analogous to the confidence interval? So if I had three arms after T plays with, say, upper bound values of: 0.09, 0.08, and 0.05, what does that actually mean?

   - **j2kun**

     October 10, 2014 at 10:32 am • Reply

     It depends on your goal. In practice, often the goal is to serve ads to users, which never ends. If you’re trying to identify the single best arm, then you have to add additional assumptions to find it because, even though the process will converge in the limit to a distribution with overwhelming weight on the best arm, the convergence rate will depend on the difference of the payoff of the

best arm and the second best arm, but you can use the last line of the main proof (along with a union bound over all the arms) to bound the probability that you pick a suboptimal arm. Then set that less than your threshold and solve for T.

7. **eta**

   February 26, 2016 at 6:33 am • Reply

   Why "then another trivial regret bound is \Delta T" ?

   - **j2kun**

     February 26, 2016 at 11:15 am • Reply

     In this case you never pick an optimal action (and all the penalties for suboptimal actions are the same). So you incur the most regret in every of the T rounds.

8. **nirandi**

   April 21, 2016 at 7:53 pm • Reply

   May I know, is UCB a good algorithm for adversarial settings, where an expert changes the reward in each iteration?

   - **j2kun**

     April 21, 2016 at 10:47 pm • Reply

     In practice, perhaps. In theory, no. See EXP3 and it's variations for that. e.g., https://jeremykun.com/2013/11/08/adversarial-bandits-and-the-exp3-algorithm/

9. **holdenlee**

   October 12, 2016 at 8:07 am • Reply

   Technical point: I don't think you can reduce to all the Δ_i being the same and then compare to the function ΔT without some further argument. You'll have to maximize a combined function $\min(R(\Delta_1,\ldots),T \max \Delta_i)$, or something like this.

10. **Allan**

    December 16, 2016 at 10:15 am • Reply

    Hello brother, congrats by your explanation! Is great, but I had some problems on running: appears this log error, when I run your code

    for roundinfo in ucb1(100, 1):
    print(roundinfo)

    Appears that on log:

    Traceback (most recent call last):
    File "/home/ubuntu/workspace/ex50/bin/mab.py", line 35, in
    for roundinfo in ucb1(100, 1):
    File "/home/ubuntu/workspace/ex50/bin/mab.py", line 13, in ucb1
    payoffSums[t] = reward(t,t)
    TypeError: 'int' object is not callable

    Can you help me please? Thankful!

    - **j2kun**

      December 16, 2016 at 10:24 am • Reply

Yes. The `reward` input to `ucb1` is a function that accepts two inputs: an integer representing which action was chosen, and the current time step. It produces as output the reward for that action in that round (the rewards may change over time). You can see an example input where the reward of each action is a biased random number here: https://github.com/j2kun/ucb1/blob/master/ucb1.py#L50

11. **Jing Lu**

December 23, 2016 at 12:26 pm • Reply

For the part "Theorem: Suppose UCB1 is run as above. Then its expected cumulative regret". For the second term with summation, should it be added from 1 to T instead of 1 to K?

12. **rk0519**

January 30, 2017 at 5:21 am • Reply

Can someone please explain the step \displaystyle \mathbb{E}(G_A(T)) = \mu_1 \mathbb{E}(P_1(T)) + \dots + \mu_K \mathbb{E}(P_K(T))

13. **jkyu**

July 25, 2017 at 2:35 am • Reply

Thanks For the clear explanation!
Why "any round in which we play an action j, it must be that a(j, T+1) = a(j,T)" ?
I think a will decrease in any round we play an action j, as the equation shows:
a = a(j,T) = \sqrt{2 \log(T) / n_j}

14. **generalrincewind**

September 14, 2017 at 6:22 am • Reply

By the Chernoff-Hoeffding inequality in this article, did you leave out the factor of two and the absolute value calculation, or is P(X-u > t) = P(X – u < t)?

- **j2kun**

  September 14, 2017 at 7:19 am • Reply

  They are not necessarily equal, but both are upper bounded by the same quantity (Hoeffding's bound has a one-sided and a two-sided version)

15. **generalrincewind**

September 20, 2017 at 1:07 pm • Reply

Is it possible to extend the UCB algorithm to a situation where the payoff of all variables are known every round? Because I've been trying to work it out myself, and the algorithm as described here just devolves into taking the variable with highest average payoff, and I'm certain that can't be the best method.

16. **generalrincewind**

September 21, 2017 at 9:32 pm • Reply

…this is the contextual bandit problem.

17. **genrincewind**

October 5, 2017 at 3:07 pm • Reply

So, but tangential, but: I've been trying to apply the UCB algorithm as described here to cryptocurrency trading,using the same mechanism you describe in your stocks post, along with a few other alternate metrics, and I've discovered that in that context the strategy that works the best by far is to simply have your metric be the average of the last three payoffs of a certain currency, it outstrips

the UCB algorithm by a factor of 2.
Now that seemed to indicate that the UCB algorithm is not the best algorithm to use in that cryptocurrency situation. I then tried to implement the Exp3 algorithm: Same result. Simply averaging the last three payoffs led to double the payoff of the Exp3 algorithm.
Now thus I have to ask: What other alternative algorithms exist for the bandit trading situation where all information is known? Simply taking the average works, but I'm sure there must be alternatives.

- **j2kun**

  October 5, 2017 at 3:19 pm • Reply

  Check out this survey for a relatively exhaustive list of the different models and algorithms: https://arxiv.org/abs/1204.5721 .

  One thing to note is that the analysis of these algorithms doesn't take into consideration constant-factor fluctuations in regret; they're primarily concerned with asymptotic performance as the number of possible actions and the number of rounds grows. So there could easily be a trivial tweak to UCB1/Exp3 that ends up doubling or tripling its payoff, but researchers would not care. As one proposal, try the UCB/Exp3 update using the average of 3 for each update. On top of that, the people who design these algorithms primarily care about what can be proved, and what insights can be gained from those proofs. Average–of-three might do better, but probably does not have any good guarantees. I personally think guarantees are important when most financial trading is about measuring and hedging risk.

  It's also important to note that cryptocurrency fluctuations are far from independent trials, as UCB posits, and probably not adversarial either.

  If you have more detailed notes about your experiments, I'd be interested to read them.

  - **genrincewind**

    October 6, 2017 at 1:47 pm

    I'll see if I can do a write-up. But just for interest's sake: The whole idea of the averaging the last three payoffs came from assuming all winning currencies are because of somebody pumping them, i.e. all crypto is just one massive pump and dump(you probably already know what a pump n dump is, it's where you artificially deflate a currency by buying a lot of it so that you can sell it at an inflated price),, and therefore the best currencies to buy in are those that are going up steadily fastest, since that indicates that somebody is pumping that currency. That assumption would lead one to saying: max payoff previous round = max payoff next round? But that strategy gives worse payoff then UCB or EXP, which indicates that crypto has still an element of randomness. Therefore you need to average it out as to filter out randomnesss, but not so much that you can;t detect when a currency is rocketing. Therefore average of 3 is a good compromise.

  - **genrincewind**

    October 13, 2017 at 11:16 pm

    So, I've been working through the survey, and part of the stochastic bandit chapter baffles me, namely: in the restatement of the pseudoregret why do they take the mean of the mean? And how did they get that from the original equation?

https://math.stackexchange.com/questions/2470951/where-does-the-expected-value-in-the-restatement-of-the-pseudoregret-come-from
Stack Overflow question concerning the question^

- **genrincewind**

  October 27, 2017 at 1:14 pm

  So, no answer? Dangit, I'd really like to know :-{

18. **brian braun**

    February 8, 2018 at 6:05 pm • Reply

    Hi
    Just stumbled across your code from wikipedia. I ran it and produced the expected results. My question is does the ucb1 method only show the regret is there a way of extracting the best action that contributed the most to the cummulative regret.

    Thanks

    - **j2kun**

      February 8, 2018 at 6:12 pm • Reply

      Yes, you can see on this line of code that the ucb algorithm produces, at each step, the chosen actions, the reward/regret, and the current set of confidence bounds it assigns to each action:

      https://github.com/j2kun/ucb1/blob/master/ucb1.py#L56

      At the end of the loop one could print out the ucbs list to see the final values for each action.

19. **Matt Wheeler**

    February 28, 2018 at 12:59 pm • Reply

    Hi Jeremy, I've implemented a simulation using your code as a base (https://github.com/datavizhokie/UCB1/blob/master/ucb1_simulation.py). My major question is why you set BestAction = 0? Is this saying the 0th arm is assumed to be the best action, or does this have a more mathematical reasoning? I noticed that if I randomize the BestAction as int in {0:numActions}, then my regret turns negative (no ideal!)

    - **j2kun**

      February 28, 2018 at 1:11 pm • Reply

      Hi Matt,

      You will notice that the "bestAction" part of the code is inside a test called "simpleTest". There I set the reward-generating functions to be uniform random variables with a biased depending on the index.

      biases = [1.0 / k for k in range(5,5+numActions)]

      So the first index is the best action, because it has the highest bias (they taper off to 1 as the index increases). The point of this was not to seed the algorithm to know the best action ahead of time, but to compare UCB's choices against the best algorithm in hindsight (which, in this test, is to always choose the first action). That is to illustrate the regret bounds and compare them to the theorem in this post. If you change which is the best, then the regret may change sign because you're comparing UCB's actions against suboptimal actions.

Hope that helps!

- **Matt Wheeler**

  February 28, 2018 at 2:47 pm

  Jeremy, thanks for the detailed response. It makes sense that you have to set the biases so that there is an priori optimal action. However, this is not the case in many practical multi-arm bandit scenarios. Do you know of a way to predict regret without designation of biases per action?

- **j2kun**

  February 28, 2018 at 3:05 pm

  If you get to observe payoffs that occur for actions you did not choose in retrospect (e.g., stock market price movements), then you can keep track of all payoffs for all actions at every time step, and compute the best single action at the end to get regret.

  If you can't view payoffs of unchosen actions (indeed, this is the point of explore/exploit), then you can't measure regret. This is why the theorem is useful, because you know that this immeasurable quantity is bounded (assuming the model hypotheses hold, which they don't in many cases).