

Bayesian Bandits - optimizing click throughs with statistics (https://www.chrisstucchio.com/blog/2013/bayesian_bandit.html)

Mon 08 July 2013 [statistics \(https://www.chrisstucchio.com/tag/statistics.html\)](https://www.chrisstucchio.com/tag/statistics.html) / [bayesian reasoning \(https://www.chrisstucchio.com/tag/bayesian-reasoning.html\)](https://www.chrisstucchio.com/tag/bayesian-reasoning.html) / [bandit algorithms \(https://www.chrisstucchio.com/tag/bandit-algorithms.html\)](https://www.chrisstucchio.com/tag/bandit-algorithms.html)

Get notified of new posts

Great news! A murder victim has been found. No slow news day today! The story is already written, now a title needs to be selected. The clever reporter who wrote the story has come up with two potential titles - "Murder victim found in adult entertainment venue" and "Headless Body found in Topless Bar". (The latter title is one I've shamelessly stolen from the NY Daily News.) Once upon a time, deciding which title to run was a matter for a news editor to decide. Those days are now over - the geeks now rule the earth. Title selection is now primarily an algorithmic problem, not an editorial one.

One common approach is to display both potential versions of the title on the homepage or news feed, and measure the Click Through Rate (CTR) of each version of the title. At some point, when the measured CTR for one title exceeds that of the other title, you'll switch to the one with the highest for all users. Algorithms for solving this problem are called [bandit algorithms \(https://en.wikipedia.org/wiki/Multi-armed_bandit\)](https://en.wikipedia.org/wiki/Multi-armed_bandit).

In this blog post I'll describe one of my favorite bandit algorithms, the Bayesian Bandit, and show why it is an excellent method to use for problems which give us more information than typical bandit algorithms.

Unless you are already familiar with Bayesian statistics and beta distributions, I strongly recommend reading the [previous blog post \(/blog/2013/bayesian_analysis_conversion_rates.html\)](/blog/2013/bayesian_analysis_conversion_rates.html). That post provides much introductory material, and I'll depend on it heavily.

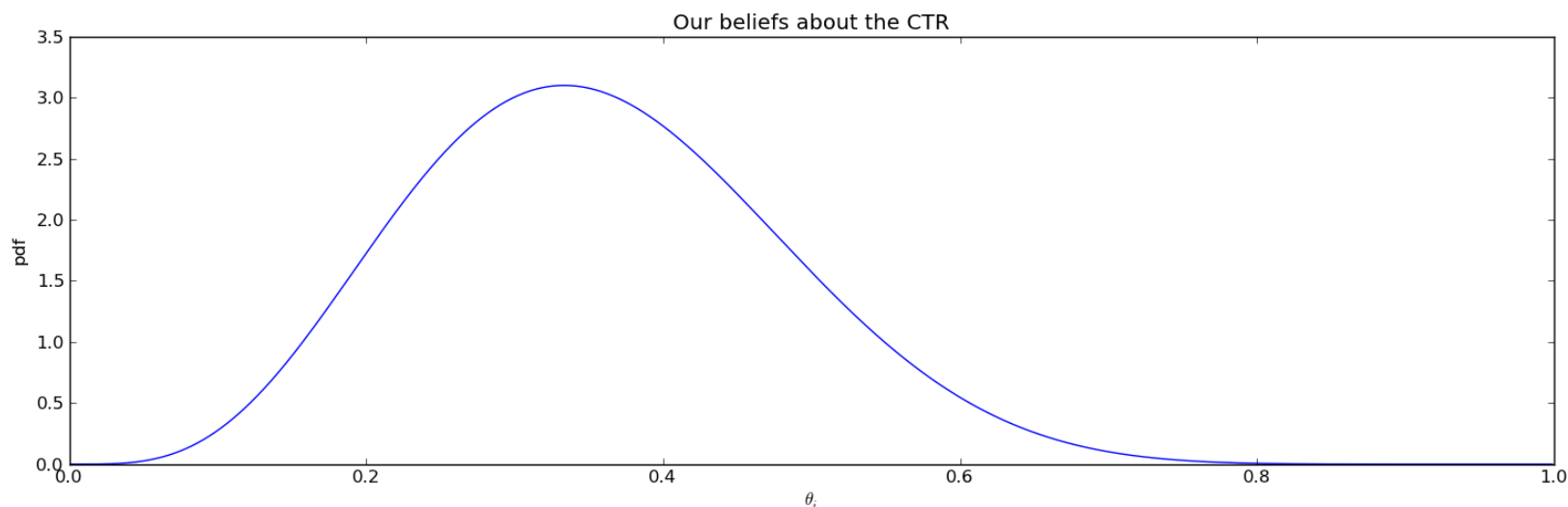
The problem to be solved, and the underlying model

Ultimately the problem we want to solve is the following. Consider an article being published on a website. The author or editor has come up with several possible titles - "Murder victim found in adult entertainment venue", "Headless Body found in Topless Bar", etc. We want to choose the title with the best click through rate (CTR). Let us represent each CTR by θ_i - i.e., θ_i is the true probability that an individual user will click on the i-th title. As a simplifying assumption, we assume that these rates θ_i do not change over time. It is important to note that we don't actually know what θ_i is - if we did, we could simply choose i for which θ_i was largest and move on.

The goal of the bandit algorithm is to do the following. To begin with, it should display all possible titles to a random selection of users, and measure which titles are clicked on more frequently. Over time, it will use these observations to infer which articles have the higher CTR. Then, once the estimation of the CTR becomes more precise, it will preferentially display articles with the higher CTR.

The Bayesian Approach

In the model described above, we have N possible story titles, each of which has a click through rate θ_i . Unfortunately we do not know what θ_i is. As the astute reader can guess from the title, we are following a Bayesian approach, so we will construct a *probability distribution* which represents our *belief* about what the actual value of θ_i is.



In the figure above, we believe that θ_i is somewhere between 0.1 and 0.7, with values of 0.3-0.4 being considerably more likely than values of 0.1-0.2 or 0.6-0.7. For those who forgot STATS 101, the area under this curve between the points a and b is the probability that θ_i lies between a and b. I.e.:

$$P(a < \theta_i < b) = \int_a^b f(x)dx$$

The basic idea behind Bayesian methods is to update our beliefs based on evidence. As we gather more data by showing different titles to other users and observing click throughs, we can incrementally narrow the width of the probability distribution.

As in all Bayesian inference, we need to choose a *prior*. The prior is something we believe to be true *before we have any evidence* - i.e., before we have shown the title to any visitors. This is just a starting point - after enough evidence is gathered, our prior will play a very minimal role in what we actually believe. Choosing a good prior is important both for mathematical simplicity, and because if your prior is accurate, you don't need as much evidence to get the correct answer.

I'll follow the approach I described in a [previous blog post \(/blog/2013/bayesian_analysis_conversion_rates.html\)](/blog/2013/bayesian_analysis_conversion_rates.html), and I'll use a [beta distribution \(https://en.wikipedia.org/wiki/Beta_distribution\)](https://en.wikipedia.org/wiki/Beta_distribution) as the prior:

$$P(\theta_i = x) = \frac{x^{\alpha_i-1}(1-x)^{\beta_i-1}}{B(\alpha_i, \beta_i)} \equiv f_{\alpha_i, \beta_i}(x)$$

The parameters $\alpha_i, \beta_i > 1$ are the prior parameters. One reasonable choice is $\alpha_i = \beta_i = 1$, which amounts to the uniform distribution on $[0, 1]$. What this means is that we are assuming that all possible values of θ_i are equally likely. Depending on the circumstances (which I'll explain shortly), we might want to choose other possible values.

Updating our beliefs

Now we address the question of using evidence.

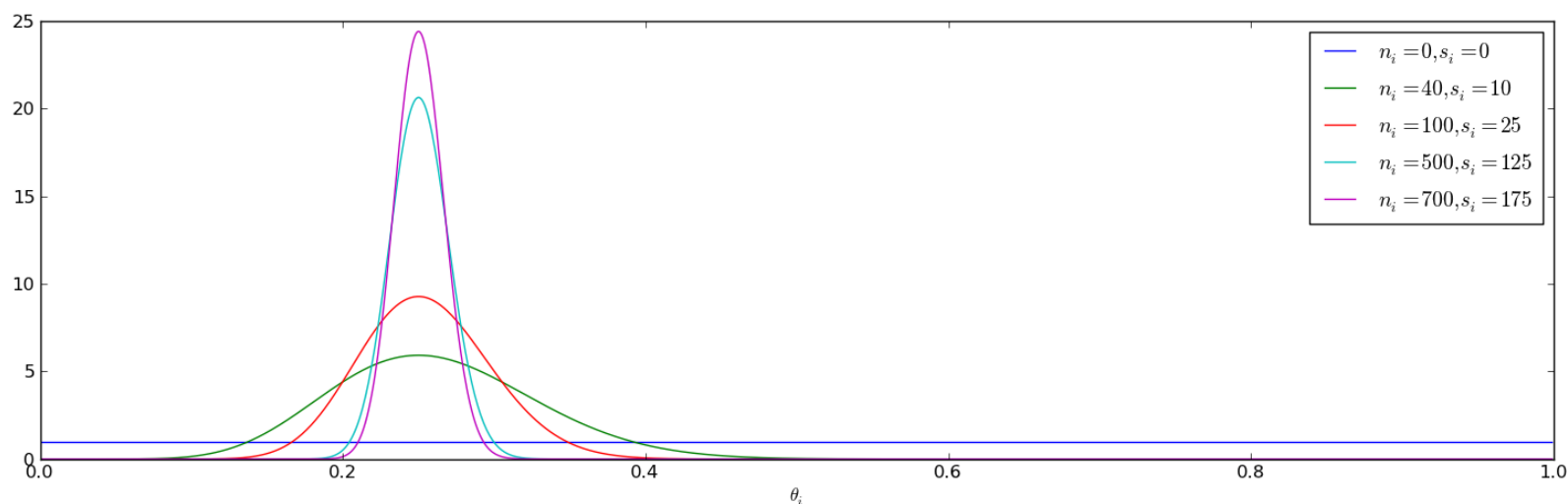
After showing title i to n_i visitors, we have observed that s_i of them have actually clicked on the title. We now want to compute the *posterior* distribution, which is to say the distribution that represents our beliefs *after* we have evidence.

I did a little bit of algebra [previously \(/blog/2013/bayesian_analysis_conversion_rates.html\)](/blog/2013/bayesian_analysis_conversion_rates.html), in which I showed that if the prior is $f_{\alpha_i, \beta_i}(\theta_i)$, then the posterior distribution is:

$$\text{posterior} = f_{\alpha_i + s_i, \beta_i + n_i - s_i}(\theta_i)$$

The key idea here is that to update our probability distribution describing θ_i , we need only update the parameters of our beta distribution.

So what does this mean in practice? As we run more experiments, our probability distribution on where θ_i lives becomes sharper:



Before we run any experiments, θ_i could be anything (as represented by the blue line). Once we have run 700 experiments, yielding 175 click throughs, we are reasonably confident that θ_i lives roughly between 0.2 and 0.3.

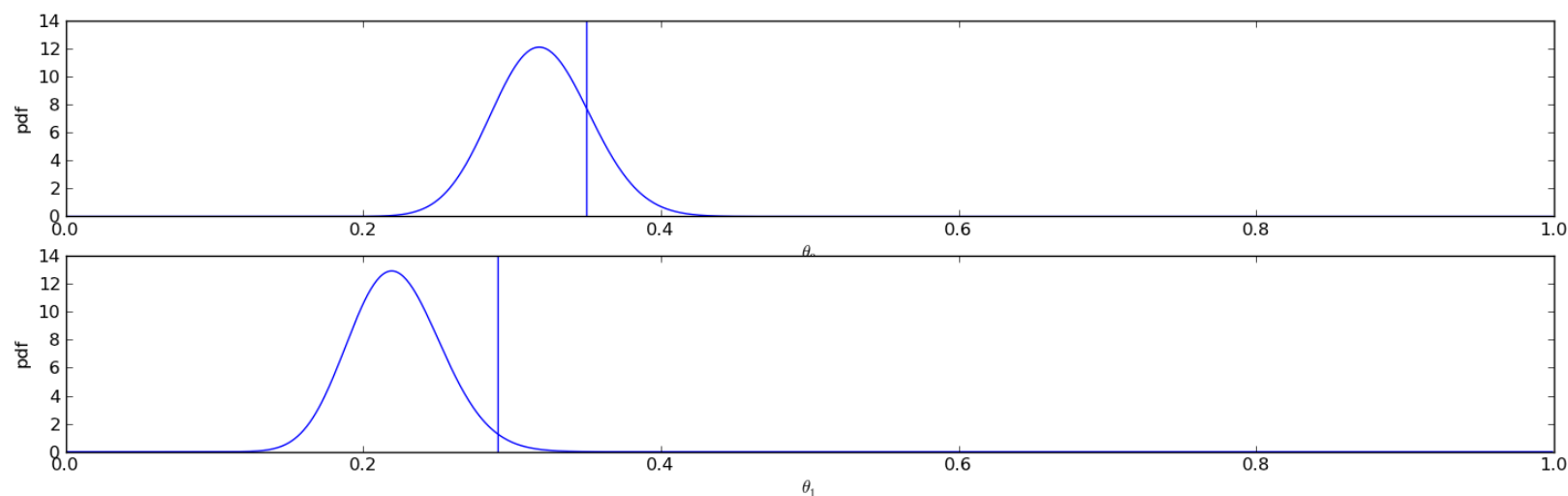
What we've done so far is figured out how to estimate what our click through rates actually are based on empirical evidence. But that doesn't actually give us a method of optimizing them yet.

Optimizing click throughs

Now that we have a method of representing our beliefs about CTRs, it is useful to construct an algorithm to identify the best ones. There are many popular choices - I've written about the [UCB Algorithm \(/blog/2012/bandit_algorithms_vs_ab.html\)](/blog/2012/bandit_algorithms_vs_ab.html) before, and I consider it a good choice.

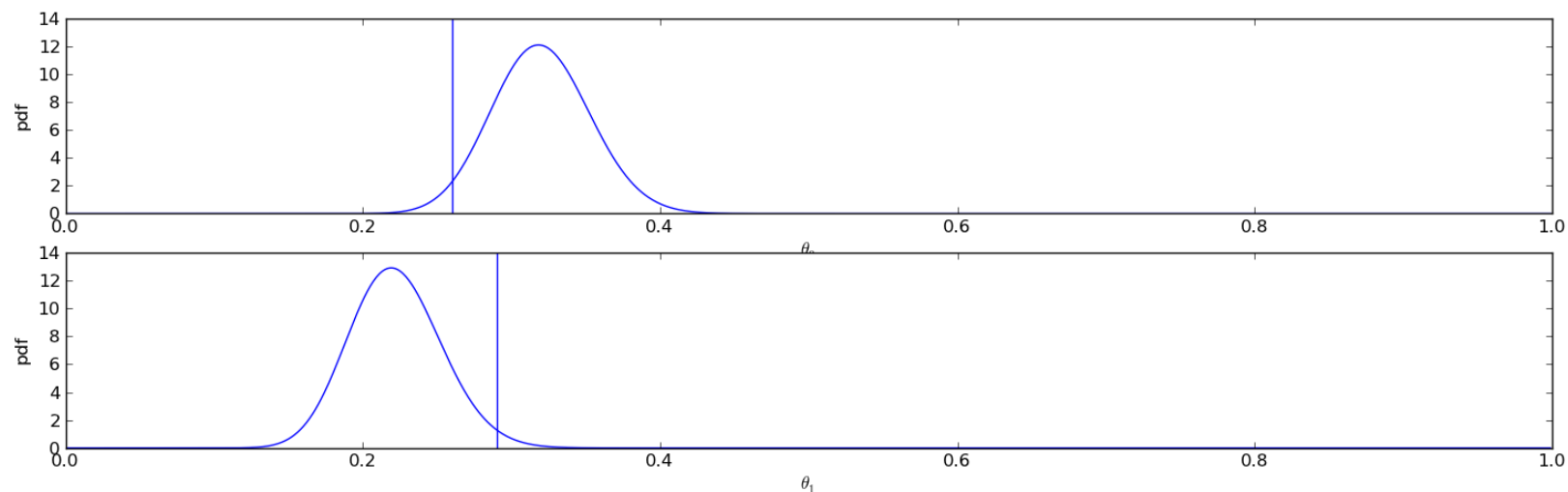
But my new favorite method is a [Monte Carlo \(https://en.wikipedia.org/wiki/Monte_Carlo_method\)](https://en.wikipedia.org/wiki/Monte_Carlo_method) method which I'll describe now.

The ultimate goal of the bandit algorithm is to display to the user whichever title has the highest CTR. One method of estimating the CTRs of the articles is to *sample* the posterior distribution. I.e., suppose we have two possible titles, from which we have drawn $n_0 = 200, s_0 = 64$ and $n_1 = 180, n_2 = 40$. Then one possible set of samples we might observe is this:



For title 0, our sample of θ_0 has worked out to be 0.35, while our sample of θ_1 is only 0.28. Since $\theta_0 = 0.35 > \theta_1 = 0.28$, we will display title 0 to the user.

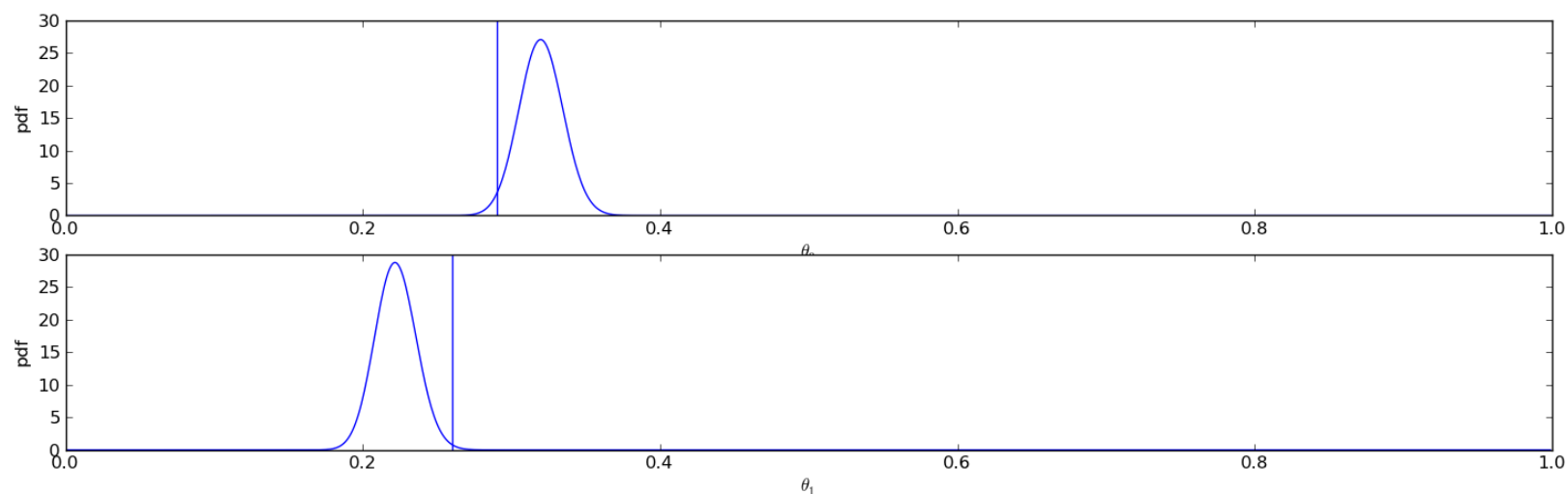
However, there was no guarantee that things worked out this way. It was possible, although less likely, that θ_1 could come out larger than θ_0 :



In this case, we would have displayed title 1 to the user rather than title 0.

The net result is that for overlapping probability distributions, we will display the title with the larger *expected* CTR the majority of the time. But occasionally, we will draw from the other distributions simply because it is within the realm of possibility that they are greater.

As we gather more data our probability distributions will become narrower and a clear winner will become apparent. When this occurs, we will almost surely choose the winner:



In python, the algorithm looks like this:

```

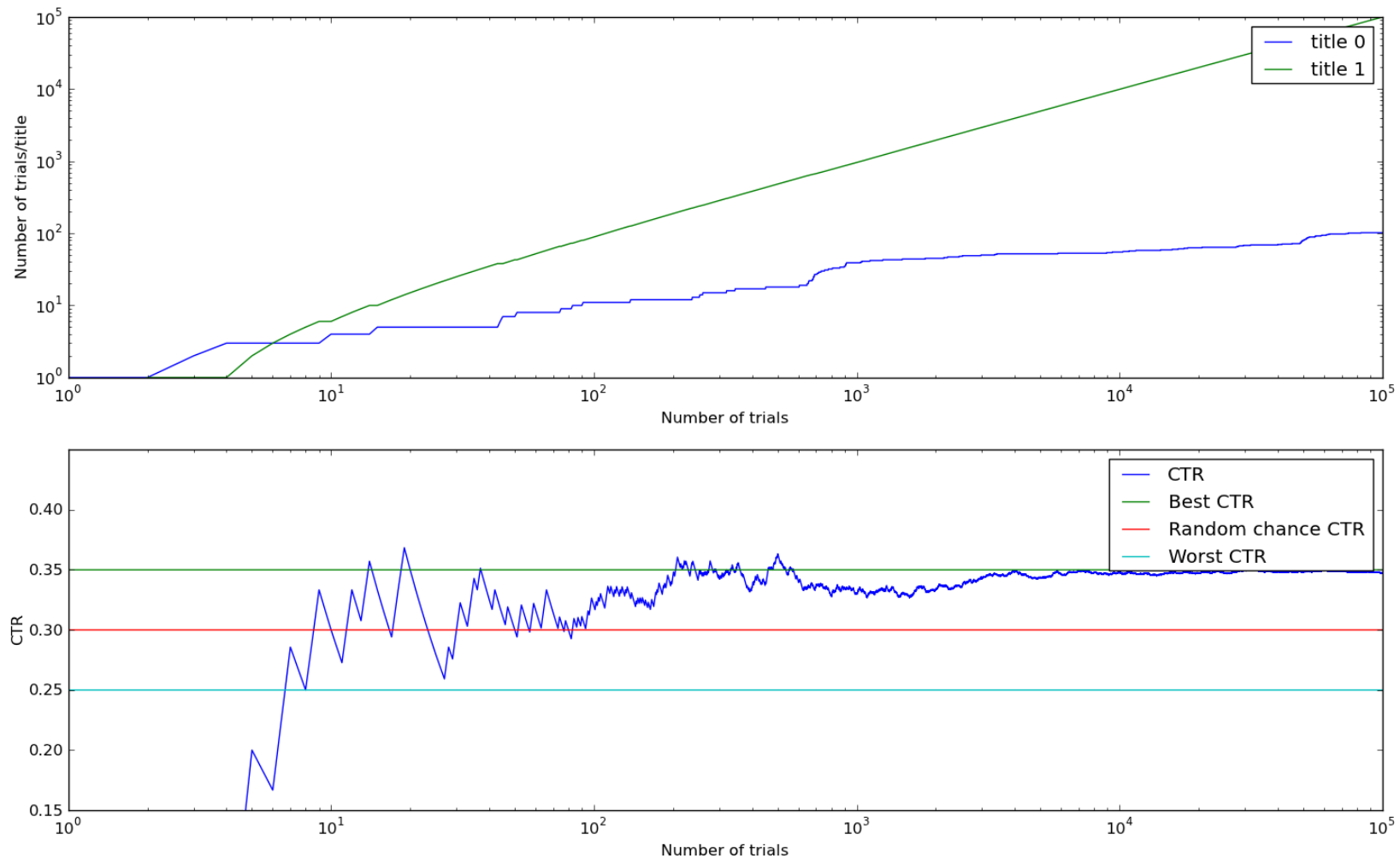
1  from numpy import *
2  from scipy.stats import beta
3
4
5  class BetaBandit(object):
6      def __init__(self, num_options=2, prior=(1.0,1.0)):
7          self.trials = zeros(shape=(num_options,), dtype=int)
8          self.successes = zeros(shape=(num_options,), dtype=int)
9          self.num_options = num_options
10         self.prior = prior
11
12     def add_result(self, trial_id, success):
13         self.trials[trial_id] = self.trials[trial_id] + 1
14         if (success):
15             self.successes[trial_id] = self.successes[trial_id] + 1
16

```

```
17     def get_recommendation(self):
18         sampled_theta = []
19         for i in range(self.num_options):
20             #Construct beta distribution for posterior
21             dist = beta(self.prior[0]+self.successes[i],
22                         self.prior[1]+self.trials[i]-self.successes[i])
23             #Draw sample from beta distribution
24             sampled_theta += [ dist.rvs() ]
25         # Return the index of the sample with the largest value
26         return sampled_theta.index( max(sampled_theta) )
```

[beta_bandit.py](https://gist.github.com/stucchio/5383149#file-beta_bandit-py)view raw (https://gist.github.com/stucchio/5383149/raw/0c4171e4192be58410e6afee2291700c243d9181/beta_bandit.py)(https://gist.github.com/stucchio/5383149#file-beta_bandit-py) hosted with ❤ by GitHub (<https://github.com>)

The results of this algorithm are exactly what any good bandit algorithm should do. I ran the following simulation, giving the beta bandit two titles - title 0 had a CTR of 0.25, title 1 had a CTR of 0.35. To start with, both titles were displayed to the user with roughly equal probability. Over time, evidence accumulated that title 1 was considerably better than title 0. At this point the algorithm switched to displaying primarily title 1, and the overall CTR of the experiment converged to 0.35 (the optimal CTR).

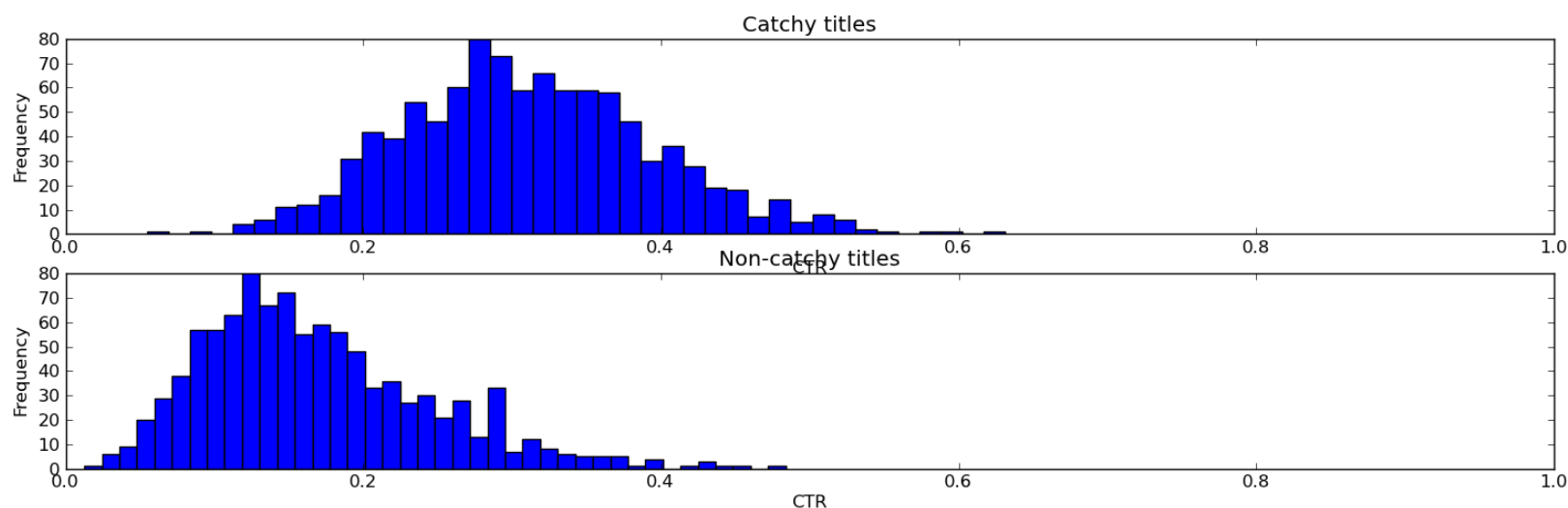


Source code to generate this graph is available here (https://gist.github.com/stucchio/5383015#file-beta_bandit_test-py). This method is called Thompson Sampling (https://en.wikipedia.org/wiki/Thompson_sampling) and is a fairly popular method in Bayesian AI techniques. For the remainder of this post, I'll call this method the Bayesian Bandit.

Incorporating common sense

Anyone with common sense is now scoffing at the geekiness embodied in this post. Even before using statistics, it was fairly obvious that "Headless Body found in Topless Bar" was going to beat "Murder victim found in adult entertainment venue". The former just sounds catchier and any good editor would run with it.

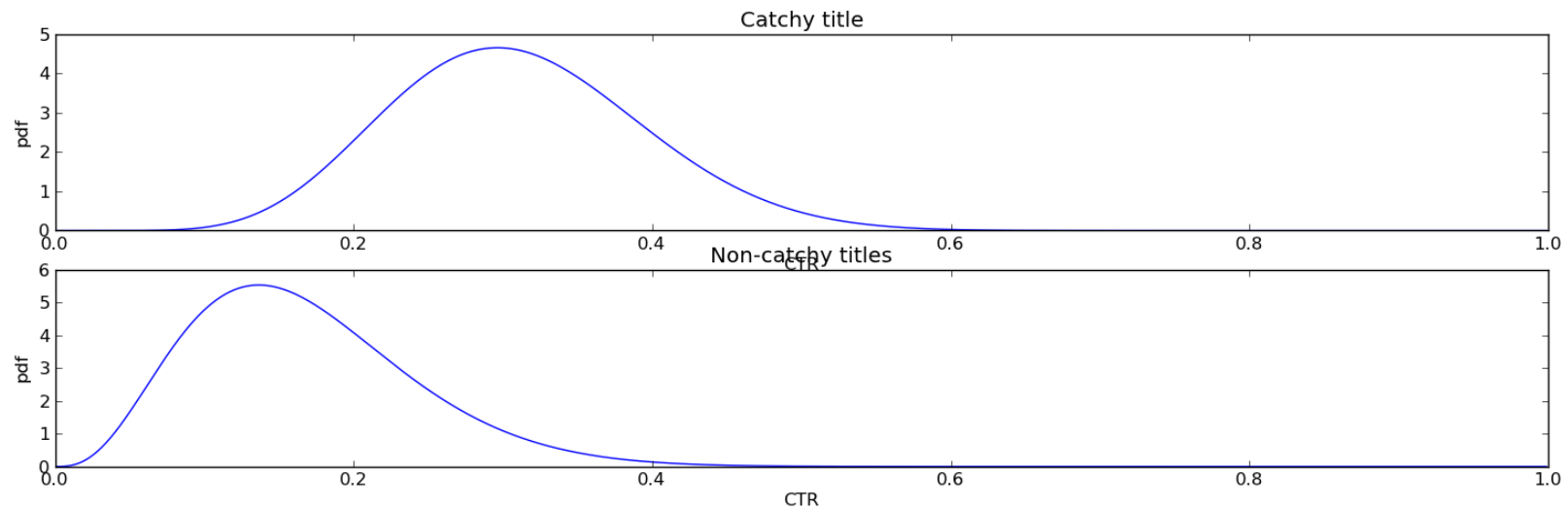
The wonderful thing about Bayesian methods is that we can modify them to take into account our prior knowledge. Suppose we believe editors intuition is a real thing - can we quantify it? Certainly. We can do this with a fairly simple experiment. We require editors to rate a collection of titles as "catchy" or "not catchy", run them on the site, and then measure the CTR of the "catchy" and "not catchy" samples. Suppose we did such an experiment, and observed the following aggregate results:



This isn't a solid win for the editor - some catchy titles have low CTRs, and some boring titles have good CTRs. But nevertheless, it's better for a story to have catchy title than not.

What we want to do is incorporate this information into our bandit algorithm. The beauty of a Bayesian method is that it gives you a clear and meaningful place to plug this information in, namely the prior. In contrast, for many other methods (e.g., [UCB \(/blog/2012/bandit_algorithms_vs_ab.html\)](http://www.chrisstucchio.com/blog/2012/bandit_algorithms_vs_ab.html)) it's somewhat difficult to do this - there is no obvious parameter to tune as a result of our prior empirical data.

The first step is to fit a theoretical distribution to the empirical data. Due to the fact that I chose the "empirical" (i.e., made up) data to be very nice, a beta distribution fits well [1] - specifically beta distributions with $(\alpha_0, \beta_0) = (9, 20)$ and $(\alpha_1, \beta_1) = (4, 20)$.



Then the only modification needed to the algorithm is to plug these variables into the prior:

$$P(\theta_0 = x) = \frac{x^{9-1}(1-x)^{20-1}}{B(9, 20)}$$

$$P(\theta_1 = x) = \frac{x^{4-1}(1-x)^{20-1}}{B(4, 20)}$$

Everything else remains unchanged. In terms of modifications to source code, this is only a very small change to the previous code - an implementation can be found [here \(https://gist.github.com/stucchio/5478642\)](https://gist.github.com/stucchio/5478642).

Empirics of including priors

To measure the benefits of incorporating priors into the Bayesian bandit, I ran some numerical experiments, the source code of which is available in this [github gist \(https://gist.github.com/stucchio/5478642\)](https://gist.github.com/stucchio/5478642). The methodology was the following.

To compare the Bayesian bandit with priors to that without, I drew a pair (θ_0, θ_1) from the prior distribution given above. For each pair, I then ran the Bayes Bandit with and without priors for this pair theta, for k trials. This is modelling the scenario that we have k page views on our homepage, and we can only leave a story on the homepage for 1 day.

I then repeated the experiment for 1000 different possible days, or equivalently for 1000 different pairs of (θ_0, θ_1) . I then computed the average gain per page view over all trials and all days.

The result is the following. If we get 50 page views/day (i.e., the Bayes Bandit has very little data to use), the prior gives us a big gain. Without prior knowledge, the Bandit achieved a gain of 0.3749 on average, whereas the bandit with prior knowledge achieved a gain of 0.4274. If we run 150 experiments, the Bayes Bandit improves significantly - it achieves a gain of 0.40. If we run 300 experiments, the Bayes Bandit improves to 0.4146, while the bandit with priors improves to 0.4296. If we get 1000 page views/day, the Bayes Bandit improves to 0.4211, while the bandit with priors gains 0.4249.

The net result is the following - incorporating priors into the Bayes Bandit is an excellent way to improve your results when you don't have a large number data points to use to train the bandit. If you have a lot of data points, you don't need strong priors (but they still help a little).

Conclusion

Bandit algorithms are a great way of optimizing many factors of your website. There are many good options - I've written about [UCB \(/blog/2012/bandit_algorithms_vs_ab.html\)](/blog/2012/bandit_algorithms_vs_ab.html) before and consider it a great choice. But if you have other information you want to include, consider using the Bayesian Bandit. It's simple to implement, straightforward to use, and very importantly it's also straightforward to extend.

It's also important to note that the theoretical properties of the Bayesian Bandit (namely logarithmic regret) have been proven (<http://arxiv.org/pdf/1111.1797.pdf>). So asymptotically, you lose nothing by using it. There are also attempts at constructing a [Bayesian UCB \(http://jmlr.csail.mit.edu/proceedings/papers/v22/kaufmann12/kaufmann12.pdf\)](http://jmlr.csail.mit.edu/proceedings/papers/v22/kaufmann12/kaufmann12.pdf) algorithm - I don't currently understand it well enough to comment.

I've written other articles related to this post. I have one post [comparing bandit algorithms to a/b testing \(/blog/2012/bandit_algorithms_vs_ab.html\)](/blog/2012/bandit_algorithms_vs_ab.html). I also I wrote about [measuring a changing conversion rate \(/blog/2013/time_varying_conversion_rates.html\)](/blog/2013/time_varying_conversion_rates.html), which provides an alternate algorithm for computing the posterior distribution if your conversion rate is not constant.

[1] If a single Beta distribution doesn't fit, one can also use a convex combination of Beta distributions. The math works out just as nicely.

P.S. After I published this blog post, [this related article \(http://camdp.com/blogs/multi-armed-bandits\)](http://camdp.com/blogs/multi-armed-bandits) was also pointed out to me, as was this [online simulation \(https://learnforeverlearn.com/bandits/\)](https://learnforeverlearn.com/bandits/) of the Bayesian bandit.

Subscribe to the mailing list

Email Address

Subscribe

Comments

Sponsored Links

Man Who Called NASDAQ Crash Has Surprising New Prediction

Investing Outlook

Man Builds Dream House For \$15,000; Look When He Opens The Door And Reveals The Inside

IcePop

This Photo Has Not Been Edited, Look Closer

Greeningz

How Much Should a 2 Carat Engagement Ring Cost?

Brilliant Earth

The One WD40 Trick Everyone Should Know About

Novelodge

These Vintage Sports Photos Were Okay Back In The Day, But Are Now Controversial

TieBreaker

1 Comment

Chris Stucchio's Blog

 Login ▾

 Recommend

 Tweet

 Share

Sort by Best ▾



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 

Name



James • a year ago

Question about your code here: <https://gist.github.com/stu...>

In the iteration over time steps:




```
for i in range(N):  
    choice = bb.get_recommendation()  
    trials[choice] = trials[choice]+1  
    conv = is_conversion(choice)  
    bb.add_result(choice, conv)
```

```
trials[i] = bb.trials  
successes[i] = bb.successes
```

Can you explain what the code, " trials[choice] = trials[choice]+1" does? the first dimension of trials is over the time horizon, isn't it? I'm confused why you do that, then set "trials[i] = bb.trials" later on. It seems like the " trials[i] = bb.trials" is all that's needed... It seems like you really just want to record what action has been tried in this time step... is that right?

Thanks in advance.

^ | v • Reply • Share ›

 Subscribe  Add Disqus to your siteAdd DisqusAdd  Disqus' Privacy PolicyPrivacy PolicyPrivacy

Sponsored Links

Almost Nobody Aces This Classic Cars Quiz!

QuizGriz

Take A Peek Inside Prince Harry & Meghan's Mansion

Refinance Gold

The Careful Reason Air Force One Has To Be Painted Blue

Livestly

This Photo Has Not Been Edited, Look Closer

Greeningz

These Unfortunate Photos Will Make You Think Twice Before Going On A Cruise

Greeningz

Almost Nobody Aces This Baseball Quiz!

QuizGriz

© 2019 Chris Stucchio · Powered by pelican-bootstrap3 (<https://github.com/DandyDev/pelican-bootstrap3>), Pelican (<http://docs.getpelican.com/>), Bootstrap (<http://getbootstrap.com>)

[↑ Back to top](#)