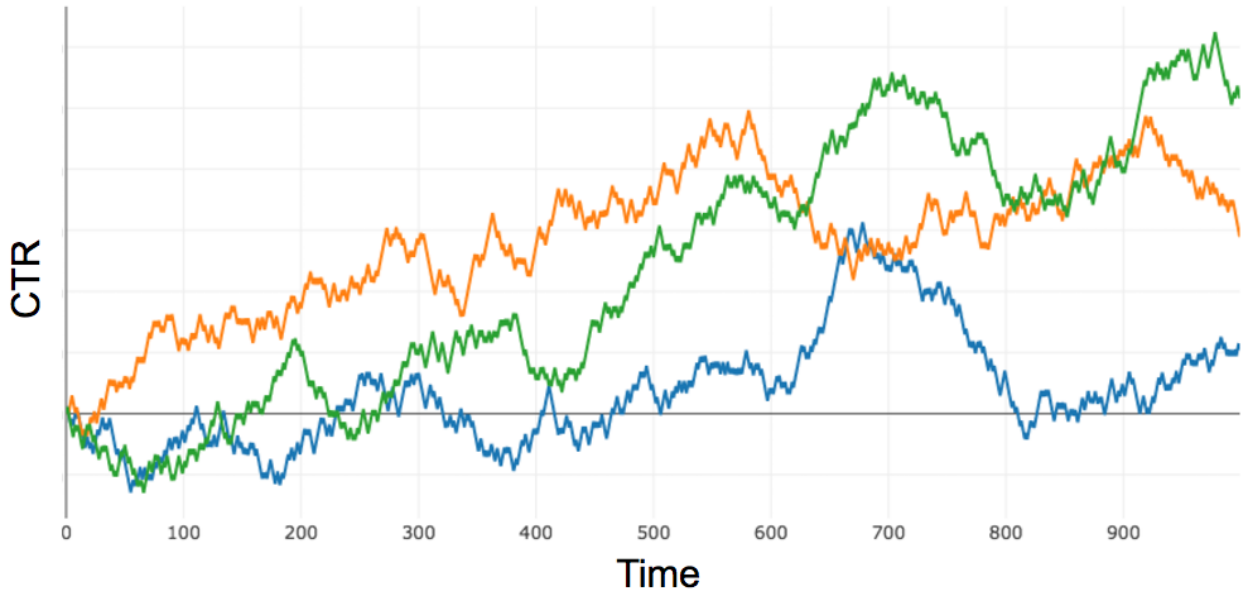
Using Bandit Algorithms on Changing Reward Rates

Introduction

One of the problems we have at System1 is updating our estimate of a feature's performance over time. Even if our initial estimate is correct, the performance of the feature could change at a future point.

Below is a graph of three page layouts (i.e. feature) and their observed click-thru rates (CTRs) (**Fig 1**). The orange layout was the best performer in the first half of the time series, but after the 600 mark the green layout became the best performer (perhaps due to a change in audience composition or preferences).

Figure 1



We estimate the performance of each feature daily to identify the highest performer, which helps us to avoid stale estimates. Splitting our system into unique populations results in tens of thousands of feature groups that we must monitor and optimize. This quickly becomes far too cumbersome for any human to track and manually update. We needed to construct an automated system in order to continue growing.

Multi-arm bandit algorithms

Bandit algorithms are designed to balance between "exploring" how features perform and "exploiting" the knowledge obtained to maximize gains. A typical bandit

implementation assumes that the underlying feature performances are constant, and thus does not give any extra weight to newer data points. In the layout example above, a bandit which assumes constant feature performance would take a long time to conclude that green outperforms orange.

Limit what the model "knows"

To control what the bandit "knows" about, we decided to use a sliding window of data. The bandit will forget about data that is older than a fixed age. This allows the bandit to reexplore each of the layouts over time.

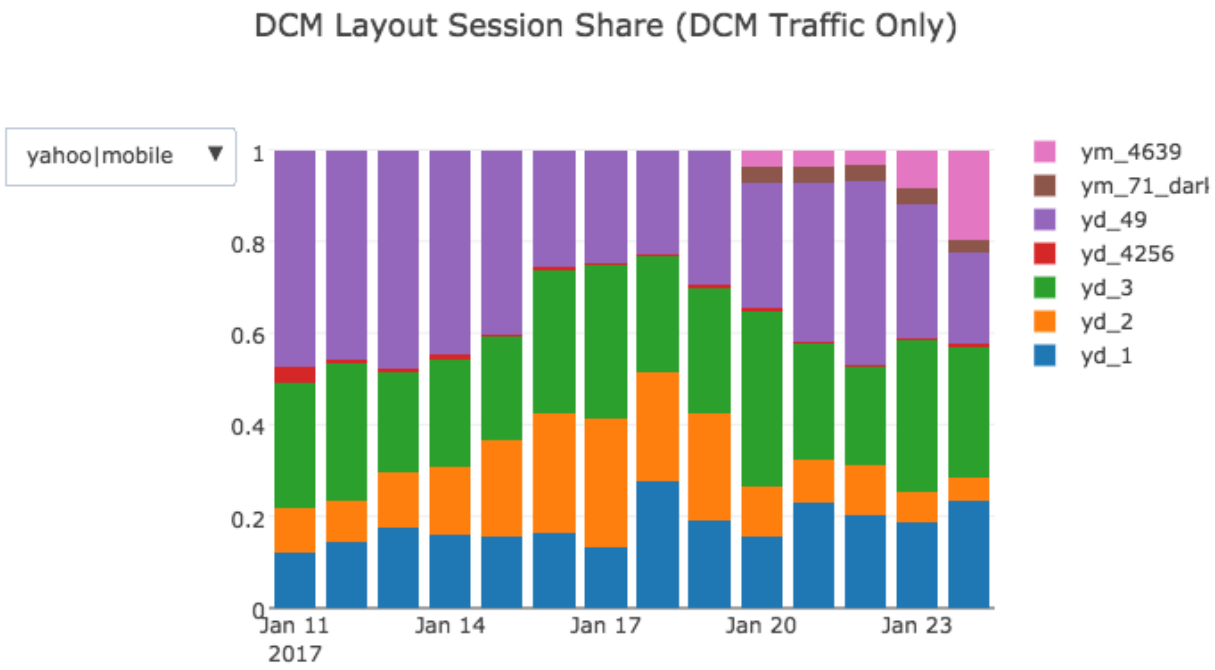
Another time-related factor we had to think about was that the value our data has in predicting future outcomes decays drastically over time. Yesterday is a much better predictor of tomorrow compared to three days ago. We added in decay to our model's calculations that would weight each day back less and less the further the model looks.

Core bandit algorithm

We chose to use the Thompson Sampling, which has several advantages:

- Exploration only occurs when there isn't enough data, as opposed to the Epsilon Greedy model which is constantly exploring at a set rate even when there is already a clear winner.
- Instead of immediately switching the bulk of traffic to the best performing layout, the bandit will tend to vary which layouts it shows more smoothly. This helps reduce shocks to our system as layout performance changes (**Fig. 2**). You can see that on Jan 20 we added "ym_4639" into our system. The model explored it for a few days before beginning to transition more traffic onto it.
- Handles situations where there isn't a clear winner by showing layouts with the same rates approximately the same amount.

Figure 2



Conclusion

In an ever-changing system, combining the limitations on what the model “knows” and the Thompson Sampling model has allowed us to continuously update our system and smoothly change to a new layout when a new winner emerges. This has saved us tremendous manual effort and when we compared our model against randomly picking the best layout have seen a lift of ~5%.

Written by Jeff Roach
Data Scientist, System1