Jan 03rd

The Mathematics of Gamification

At Foursquare, we maintain a database of 60 million venues. And like the world it represents, our database is ever-changing, with users from all over the world submitting updates on everything from the hours of a restaurant to the address of a new barbershop. To maintain the accuracy of our venue database, these changes are voted upon by our loyal Superusers (SUs) who vigilantly maintain a watchful eye over our data for their city or neighborhood.

Like many existing crowd-sourced datasets (Quora, Stack Overflow, Amazon Reviews), we assign users *points* or *votes* based on their tenure, reputation, and the actions they take. Superusers like points and gamification. It rewards diligent, hard-working SUs (which are the majority) and punishes the few malicious "bad players." But data scientists like probabilities and guarantees. We're interested in making statements like, "we are 99% confident that each entry is correct." How do we allocate points to users in a way that rewards them for behavior but allows us to make guarantees about the accuracy of our database?

At Foursquare, we have a simple, first-principles based method of resolving proposed venue attribute updates. We can gauge each Superuser's voting accuracy based on their performance on honeypots (proposed updates with known answers which are deliberately inserted into the updates queue). Measuring performance and using these probabilities correctly is the key to how we assign points to a Superuser's vote.

The Math

Let's make this more concrete with some math. Let H_0 denote the true state of the world, either 1 or -1, which we can interpret as a proposed update being true or false, respectively. We do not observe this but we know $H_0 = 1$ with a-priori probability p_0. User 1 votes H_1 (again, either 1 or -1, representing "yay" or "nay") with independent probability p_1 of agreeing with the truth H_0 and $(1 - p_1)$ of disagreeing. Bayes' Rule then gives us

$$\begin{aligned}\mathbf{P}(H_0 = 1 | H_1 = 1) &= \frac{\mathbf{P}(H_1 = 1 | H_0 = 1)\mathbf{P}(H_0 = 1)}{\mathbf{P}(H_1 = 1)} \\ &= \frac{p_0 p_1}{p_0 p_1 + (1 - p_0)(1 - p_1)} \\ &= \frac{\ell_0 \ell_1}{\ell_0 \ell_1 + 1}\end{aligned}$$

where we have written the solution in terms of the likelihood ratio $\ell_k = \ell(p_k)$ given by

$$\ell(p) = \frac{p}{1-p} \qquad \ell^{-1}(\cdot) = \frac{\cdot}{1+\cdot}\,.$$

Then we have that

$$\mathbf{P}(H_0 = 1 | H_1 = 1) = \ell^{-1}(\ell(p_0)\ell(p_1))\,.$$

In fact, it is easy to see that in the general case,

Recent Posts

- The Mathematics of Gamification
- Foursquare's new notifications and the future of contextual mobile experiences
- A chat about data science and our fun visualizations
- How we built our Model Training Engine
- Foursquare Native Auth on iOS and Android: Developers, connect your users more quickly than ever

Recent Comments

- Aledv on Say hello to FSNetworking: A Small Networking Library for iOS and Mac
- Dsawdsa on The Opportunity Gap: learn about the public schools in your neighborhood with @ProPublica and Foursquare
- Alanjim05 on The Opportunity Gap: learn about the public schools in your neighborhood with @ProPublica and Foursquare
- Orcunb on Say hello to FSNetworking: A Small Networking Library for iOS and Mac
- cargpsanddvdplayer on A Foursquare Hackathon for the new year – sign up and build the next amazing hack!

Archives

- January 2014
- December 2013
- August 2013
- July 2013
- June 2013
- May 2013
- January 2013
- December 2012
- November 2012
- October 2012
- September 2012

$$\mathbf{P}(H_0 = 1 | H_1 = h_1) = \ell^{-1}(\ell(p_0)\ell(p_1)^{h_1}) .$$

Multiplication is hard so we will define the logit or log-likelihood function

$$\text{logit} : (0, 1) \to (-\infty, \infty)$$

given by

$$\text{logit}(p) = \log(\ell(p)) = \log\left(\frac{p}{1-p}\right) \qquad \text{logit}^{-1}(\cdot) = \frac{e^{\cdot}}{1+e^{\cdot}} .$$

Then we have

$$\mathbf{P}(H_0 = 1 | H_1 = h_1) = \text{logit}^{-1}(\text{logit}(p_0) + h_1 \text{logit}(p_1)) .$$

Continuing, assume that after user 1 casts their vote, user 2 votes H_2 with an independent probability p_2 of being correct (i.e. agreeing with H_0). We can think of the posterior probability $\mathbf{P}(H_0 = 1 | H_1 = h_1)$ as our new prior and inductively repeat the above Bayesian analysis to obtain

$$\begin{aligned} \mathbf{P}(H_0 = 1 | H_1 = h_1, H_2 = h_2) &= \text{logit}^{-1}\big(\text{logit}\big(\mathbf{P}(H_0 = 1 | H_1 = h_1)\big) + h_2 \text{logit}(p_2)\big) \\ &= \text{logit}^{-1}(\text{logit}(p_0) + h_1 \text{logit}(p_1) + h_2 \text{logit}(p_2)) . \end{aligned}$$

In fact, if we have n votes $H_1, \ldots, H_n$, then we have

$$\begin{aligned} \mathbf{P}(H_0 = 1 | H_1 = h_1, \ldots, H_n = h_n) \\ = \text{logit}^{-1}\left(\text{logit}(p_0) + \sum_{k=1}^{n} h_k \text{logit}(p_k)\right) . \end{aligned} \qquad (1)$$

The Solution

The above equation suggests that we should assign s_k points or votes to user k based on

$$s_k = \text{logit}(p_k) . \qquad (2)$$

We can add up all the “yay” votes and subtract all the “nay” votes to obtain a score for the update. This score can easily be interpreted as a probability that the update is correct. We can set a certainty threshold p (e.g. $p = 99\%$) as a threshold for a desired accuracy of this edit. Then, we accept a proposed edit as soon as

$$\text{logit}(p_0) + \sum_{k=1}^{n} h_k \text{logit}(p_k) \geq \text{logit}(p) \qquad (3)$$

and reject it as soon as

$$\text{logit}(p_0) + \sum_{k=1}^{n} h_k \text{logit}(p_k) \leq - \text{logit}(p) . \qquad (4)$$

In other words, if we take $t = \text{logit}(p)$ to the the *points* threshold and $s_0 = \text{logit}(p_0)$ to be the points allocated to a new proposed edit, then (3) and (4) become

$$s_0 + \sum_{k=1}^{n} h_k s_k \geq t$$

and

$$s_0 + \sum_{k=1}^{n} h_k s_k \leq -t ,$$

which are exactly the equations for voting you would expect. But now, they're derived from math!

The Benefits

- **Efficient, data-driven guarantees about database accuracy.** By choosing the points based on a user's accuracy, we can intelligently accrue certainty about a proposed update and stop the voting process as soon as the math guarantees the required certainty.
- **Still using points, just smart about calculating them.** By relating a user's accuracy and the certainty threshold needed to accept a proposed update to an additive point system (2), we can still give a user the points that they like. This also makes it easy to take a system of ad-hoc points and convert it over to a smarter system based on empirical evidence.
- **Scalable and easily extensible.** The parameters are automatically trained and can adapt to changes in the behavior of the userbase. No more long meetings debating how many points to grant to a narrow use case.
  So far, we've taken a very user-centric view of p_k (this is the accuracy of user k). But we can go well beyond that. For example, p_k could be "the accuracy of user k's vote given that they have been to the venue three times before and work nearby." These clauses can be arbitrarily complicated and estimated from a (logistic) regression of the honeypot performance. The point is that these changes will be based on *data* and not subjective judgments of how many "points" a user or situation should get.

Some practical considerations:

- In practice, we might want a different threshold for accepting (3) versus rejecting (4) a proposed edit.
- For notational simplicity, we have assumed that a false positives and false negatives in user k's voting accuracy have the same probability p_k. In general, this is not the case. We leave it to the reader to figure the math of the general case.
- **Users like integer points.** We have to round s_k to the nearest integer. Because we can multiply linear equations like (3) and (4) by a positive constant, we can set $s_k = [\alpha \cdot \operatorname{logit}(p_k)]$ where $[\cdot]$ is the rounding function and α is a large positive constant. A large α will prevent the loss of fidelity.
- We've explained how to obtain $p_1, p_2, \ldots$ from honeypots but how do we obtain p_0, the accuracy of newly proposed updates. One way is to use the above to bootstrap those accuracies from voting: we can use this voting technique to infer the accuracy of proposals by looking at what fraction of proposed updates are accepted!
- **Bayesian Smoothing.** We assume a relatively low-accuracy prior for the accuracy of individuals. This is a pessimistic assumption that keeps new, untested users from having too much influence. It also rewards users for lending their judgment and casting votes as long as those are more accurate than our pessimistic prior. Of course, we also increase the likelihood of showing new Super Users honeypots to give them a chance to prove themselves.

–Michael Li
Data Scientist
@tianhuil

Posted in **Foursquare Engineering Blog**