2022

"HOW DO I KNOW IT WORKS?"

A technical brief on implementing effective, relevant, and adaptive messaging

aampe

"HOW DO I KNOW IT WORKS?"

IT'S A REASONABLE QUESTION.

Sometimes we hear it from a team deciding whether to give Aampe a try. Sometimes we've run push notifications for several weeks before they ask. But they always ask, and they should: **"how do we know this is working?"**

This question goes to the heart of how we evaluate messaging - the thing we call the "the *what-if?* model" in [A User Story](), the illustrated storybook we wrote to explain our algorithms.

That's me!

A User Story

A book for children, folks with lots of imagination, and people in product marketing.

aampe

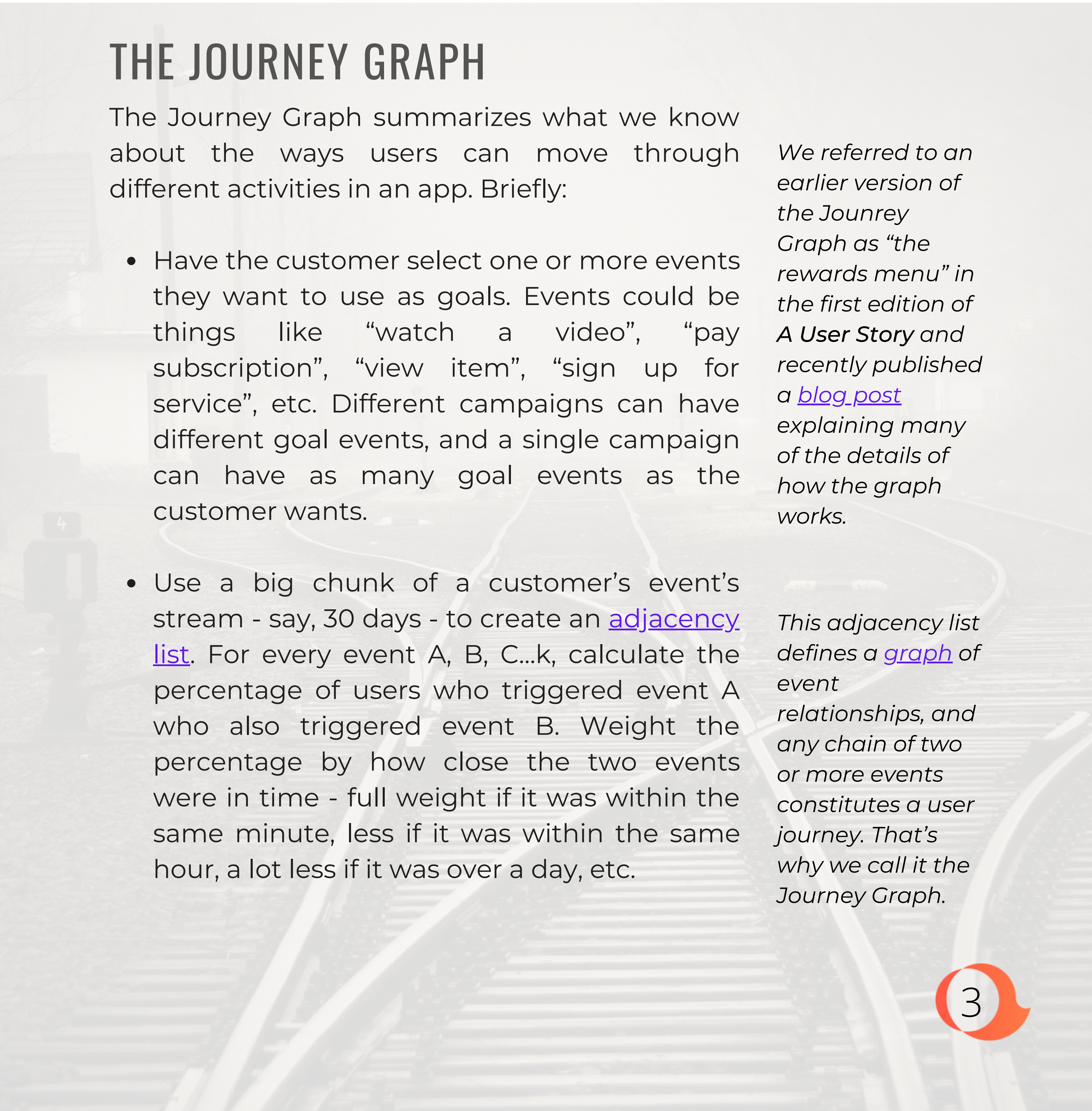
*Read **A User Story** by visiting [aampe.com/blog/a-user-story]()*

This booklet is not the storybook version. It's going to get a little technical. **We're going to explain, as the team who designed Aampe's ability to learn, why we are confident in that learning.**

AN OVERVIEW OF OUR LEARNING SYSTEM

This section delves into the weeds, touching on the technical details of each major part of our learning system. Be prepared for lots of jargon, with links to documentation or Wikipedia pages.

A high-level overview of this section exists in A User Story.



THE JOURNEY GRAPH

The Journey Graph summarizes what we know about the ways users can move through different activities in an app. Briefly:

- Have the customer select one or more events they want to use as goals. Events could be things like "watch a video", "pay subscription", "view item", "sign up for service", etc. Different campaigns can have different goal events, and a single campaign can have as many goal events as the customer wants.

We referred to an earlier version of the Jounrey Graph as "the rewards menu" in the first edition of ***A User Story*** *and recently published a blog post explaining many of the details of how the graph works.*

- Use a big chunk of a customer's event's stream - say, 30 days - to create an adjacency list. For every event A, B, C...k, calculate the percentage of users who triggered event A who also triggered event B. Weight the percentage by how close the two events were in time - full weight if it was within the same minute, less if it was within the same hour, a lot less if it was over a day, etc.

This adjacency list defines a graph of event relationships, and any chain of two or more events constitutes a user journey. That's why we call it the Journey Graph.

- For each goal event, traverse the Journey Graph to find all simple paths leading to the goal. For the sake of limiting computational complexity, limit the path traversal to some reasonable size - say 4 or 5 steps. For each path, aggregate all of the edge weights, penalizing weights that are further from the goal (a weighted harmonic mean works well). The result is a heuristic that represents the relative degree to which each event feeds into the target event. Normalize the weights so the largest weight is 1.0.

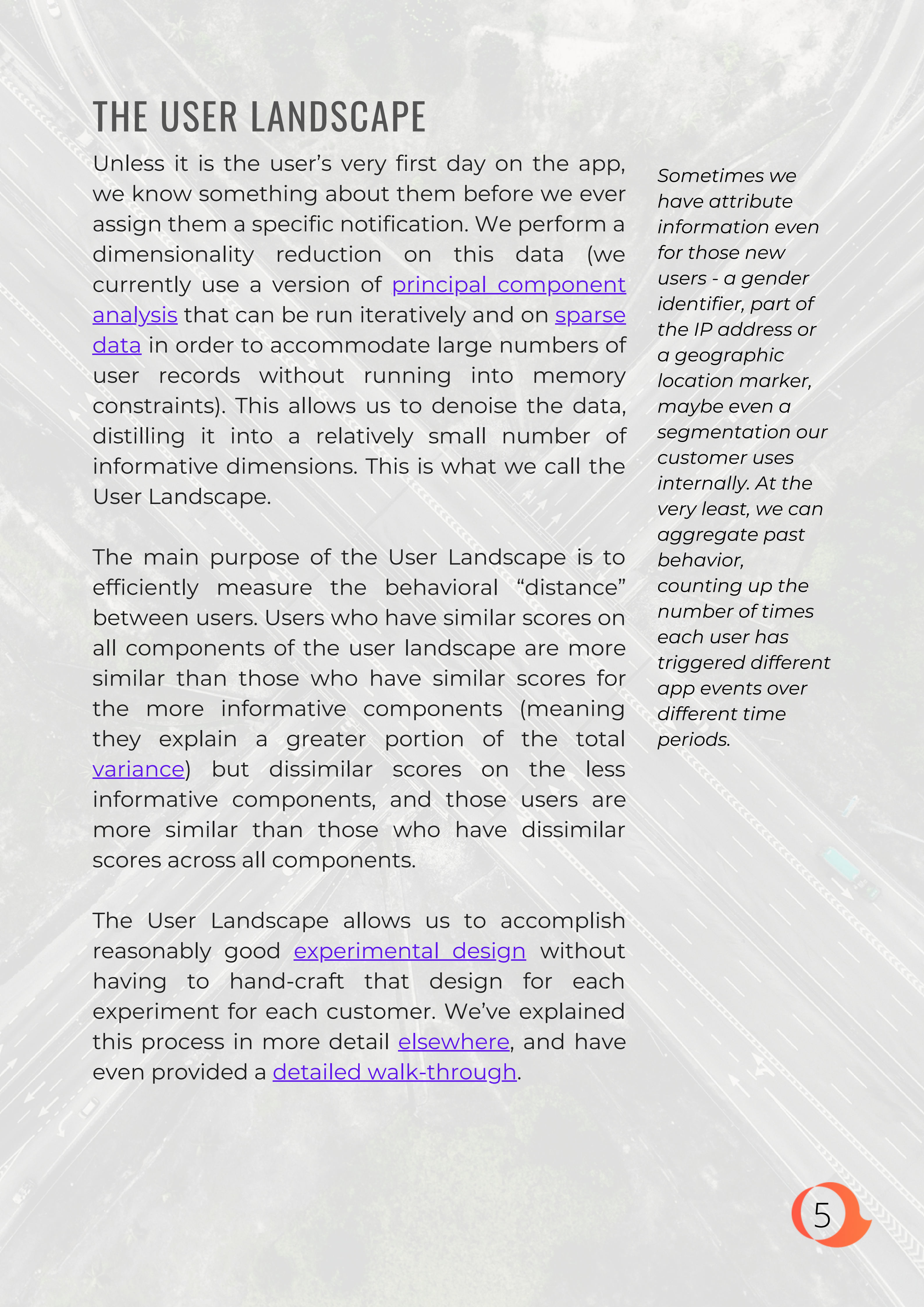
If multiple goal events have been defined, average the event weights across goals to get a single weighting scheme.

We use weights from the Journey Graph in our models. We want to focus our learning on the events that our customers think are important. If we limited to feed our models data that directly related to the goal events, we'd face two main problems:

- We'd ignore relevant behavior. Even if a user did everything users normally do to lead up to the goal behavior, but hadn't done the goal behavior itself at the time we ran the model, that would count as just as much as failure as if the user had done nothing at all.

- Often, we'd lack the data to train the model. Goal behavior is often a goal because it is relatively rare. We aren't interested in how much the entire user base did a thing - we divide the user base into smaller groups that get exposed to different messaging dynamics. Ideally, we want to see positive and negative cases in all of those subsets.

Allowing all events to count as positive behavior, but weighting how much each event really counts when training the model, allows us to avoid the two problems described here.



THE USER LANDSCAPE

Unless it is the user's very first day on the app, we know something about them before we ever assign them a specific notification. We perform a dimensionality reduction on this data (we currently use a version of principal component analysis that can be run iteratively and on sparse data in order to accommodate large numbers of user records without running into memory constraints). This allows us to denoise the data, distilling it into a relatively small number of informative dimensions. This is what we call the User Landscape.

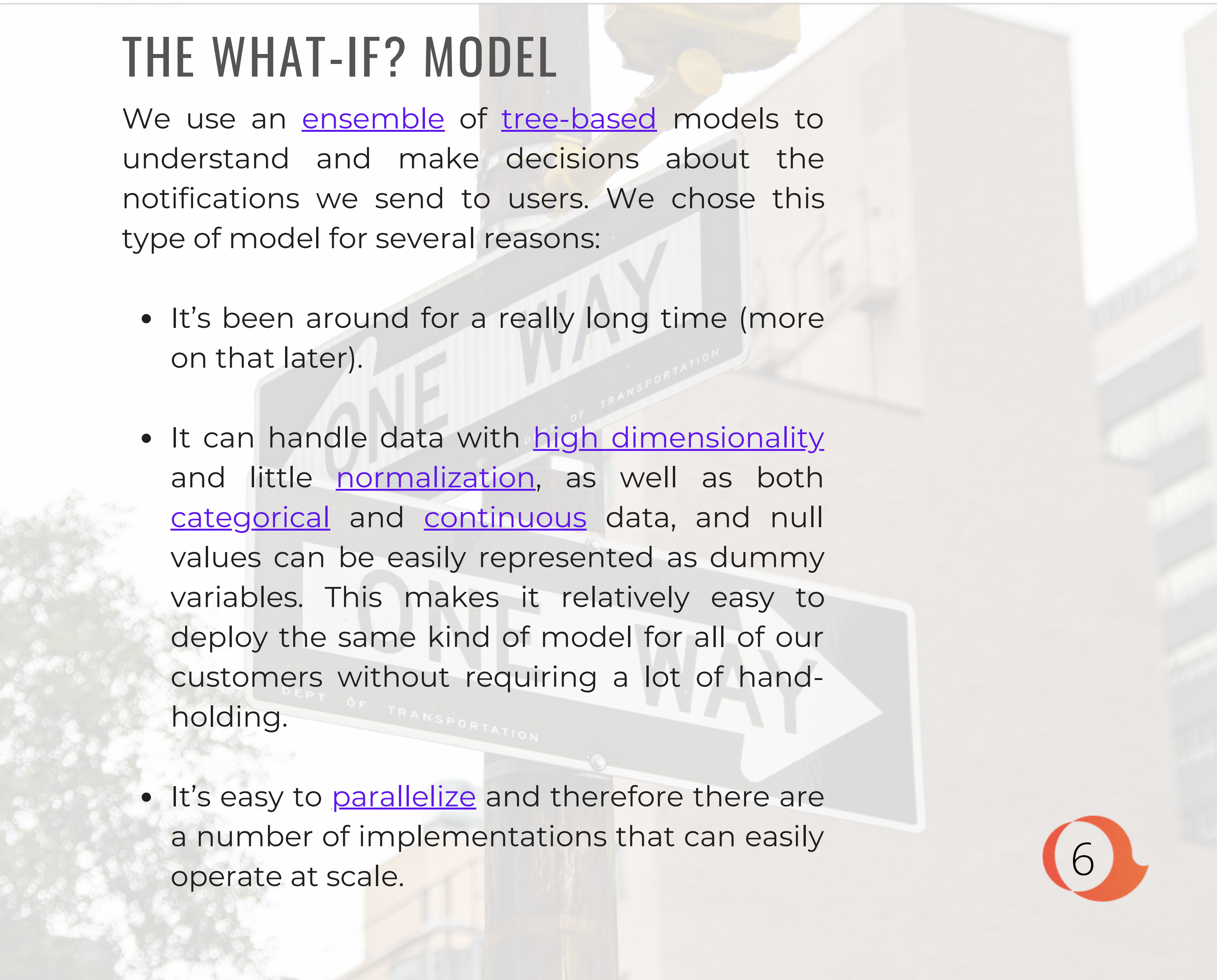
Sometimes we have attribute information even for those new users - a gender identifier, part of the IP address or a geographic location marker, maybe even a segmentation our customer uses internally. At the very least, we can aggregate past behavior, counting up the number of times each user has triggered different app events over different time periods.

The main purpose of the User Landscape is to efficiently measure the behavioral "distance" between users. Users who have similar scores on all components of the user landscape are more similar than those who have similar scores for the more informative components (meaning they explain a greater portion of the total variance) but dissimilar scores on the less informative components, and those users are more similar than those who have dissimilar scores across all components.

The User Landscape allows us to accomplish reasonably good experimental design without having to hand-craft that design for each experiment for each customer. We've explained this process in more detail elsewhere, and have even provided a detailed walk-through.

We can use the latent space that makes up the User Landscape to cluster users into relatively homogeneous groups and then use those groups to make messaging assignments. So if we want to assign, say, 35 different time increments (say, five 3-hour messaging windows per day, with a no-message period during the night, spread over all seven days of the week), then we can combine users into clusters of 35 users each and randomly assign one of each of those time increments within each cluster. That ensures that similar people get different assignments and different people get similar assignments.

It's become a truism to talk about machine learning models in terms of "garbage in, garbage out." Our User Landscape and "conditioned" assignment of messaging features ensures we don't put too much garbage into our model.



THE WHAT-IF? MODEL

We use an ensemble of tree-based models to understand and make decisions about the notifications we send to users. We chose this type of model for several reasons:

- It's been around for a really long time (more on that later).

- It can handle data with high dimensionality and little normalization, as well as both categorical and continuous data, and null values can be easily represented as dummy variables. This makes it relatively easy to deploy the same kind of model for all of our customers without requiring a lot of hand-holding.

- It's easy to parallelize and therefore there are a number of implementations that can easily operate at scale.

- It tends to achieve reasonably good results even for "small" data. That's important for smaller apps that work with us.

- It's reasonably easy to evaluate model performance using cross-validation and several well-known metrics (more on that later).

- It can implicitly handle non-linear relationships, so we don't have to investigate, discover, and manually specify interactions between variables.

- It's pretty robust against violations of assumptions and is relatively difficult to over-fit.

It's become a truism to talk about machine learning models in terms of "garbage in, garbage out." Our User Landscape and "conditioned" assignment of messaging features ensures we don't put too much garbage into our model.

We call our implementation a "*What-If?* model" because the logic of decision trees is particularly well-adapted to asking questions about what might happen under certain circumstances. All that's required is that the model have a reasonable number of examples of situations where different conditions are observed. Our conditioned assignment allows us to ensure that the model gets all the comparisons it needs to answer all the questions we want to ask.

For example: "we messaged this user on a Monday but what if we had messaged them on a Friday?", "what if we had messaged a (behaviorally) very different user on a Monday?"

A NOTE ON TECHNOLOGY CHOICES

We noted that tree ensembles have been around for a long time. We wholeheartedly endorse the position voiced in 2015 by Dan McKinley: "choose boring technology." Older technologies are much more battle-tested than newer ones. That, in and of itself, is not reason enough to choose an older technology, but it is reason enough to choose an older technology unless you can find a really compelling reason (a business reason, not just personal interest), to choose the newer technology.

The first implementation of the random forest, for example, is almost 30 years old, and decision trees are at least 10 years older

But other than that: time, money, and attention spent on cutting-edge technology is time, money, and attention not spent on innovating in other ways. As McKinley put it:

> ***"...every company gets about three innovation tokens. You can spend these however you want, but the supply is fixed for a long while.... If you choose to write your website in [insert latest javascript framework here], you just spent one of your innovation tokens.... If you choose to use service discovery tech that's existed for a year or less, you just spent one of your innovation tokens. If you choose to write your own database, oh god, you're in trouble."***

At Aampe, we care about helping apps treat their users like people - listen to them, understand what they like and don't like, and respond promptly and authentically and not so often that they get sick of hearing from you. That requires a lot of innovation. We don't need to innovate on our model implementation. We need a workhorse model that can handle the load of all of our other innovations.

One last note, just because it's a question that has come up repeatedly: "why don't you use deep learning?" You can find ten answers to that question here, five of which apply most directly to our situation:

- **Deep learning is data hungry**. Really data hungry. We don't think you should need to have a million daily active users in order to send notifications that won't bug or bore your users.
- **Transfer learning isn't robust yet**. We need to take lessons from one situation and apply those lessons to other situations. We've worked out a way to do that, and it doesn't involve deep learning.
- **It's not sufficiently transparent**. Deep learning is the most black-boxy of the black-box algorithms, which makes it harder to debug, and harder to adapt to new applications. It's not worth spending our attention on this at this time - refer to the point made earlier about innovation.
- **It mostly only works well on very stable worlds**. Deep learning can beat the pants off a human when it comes to Jeopardy or Go, but its performance is not so impressive in more organic applications.
- **It's computationally expensive**. It's not just a matter of having a lot of memory. Often deep learning requires specialized hardware to run at scale. We have no reason to make that investment at this time.

That's more than we generally like to say about deep learning, but, as we said, it comes up often. Long story short: that's not where we choose to invest our innovation tokens. Other investments are going to give us, and our customers, a better return.

Yes, deep learning is possibly the only thing that will ever get us self-driving cars, but (1) we don't have real self-driving cars yet, and (2) we aren't building self-driving cars at Aampe. We aren't dealing with computer vision or audio processing or any of the other things that deep learning is really good at.

ALTERNATIVE HYPOTHESES

In the previous section, we explained how we do what we do. In this section, we explain why what we do *works*.

Science is about defining multiple competing hypotheses (stories about what you think is going on) and then doing your level best to destroy all of those ideas. Those stories that refuse to be destroyed deserve the least amount of skepticism. **We evaluate our work through a process of story-killing.**

Recently, one of our co-founders published a post about the lazy thinking that leads many people to A/B test their way into unwarranted conclusions. Creating a "control" group and comparing it to a "test" group is not inherently scientific.

Our preferred hypothesis could be stated as: "the users we messaged did a thing because we sent each individual user the individually right message at the individually right time." There are a number of plausible alternative hypotheses:

- The model we used to make those individualized decisions was so uninformative that we actually made decisions based on random noise.

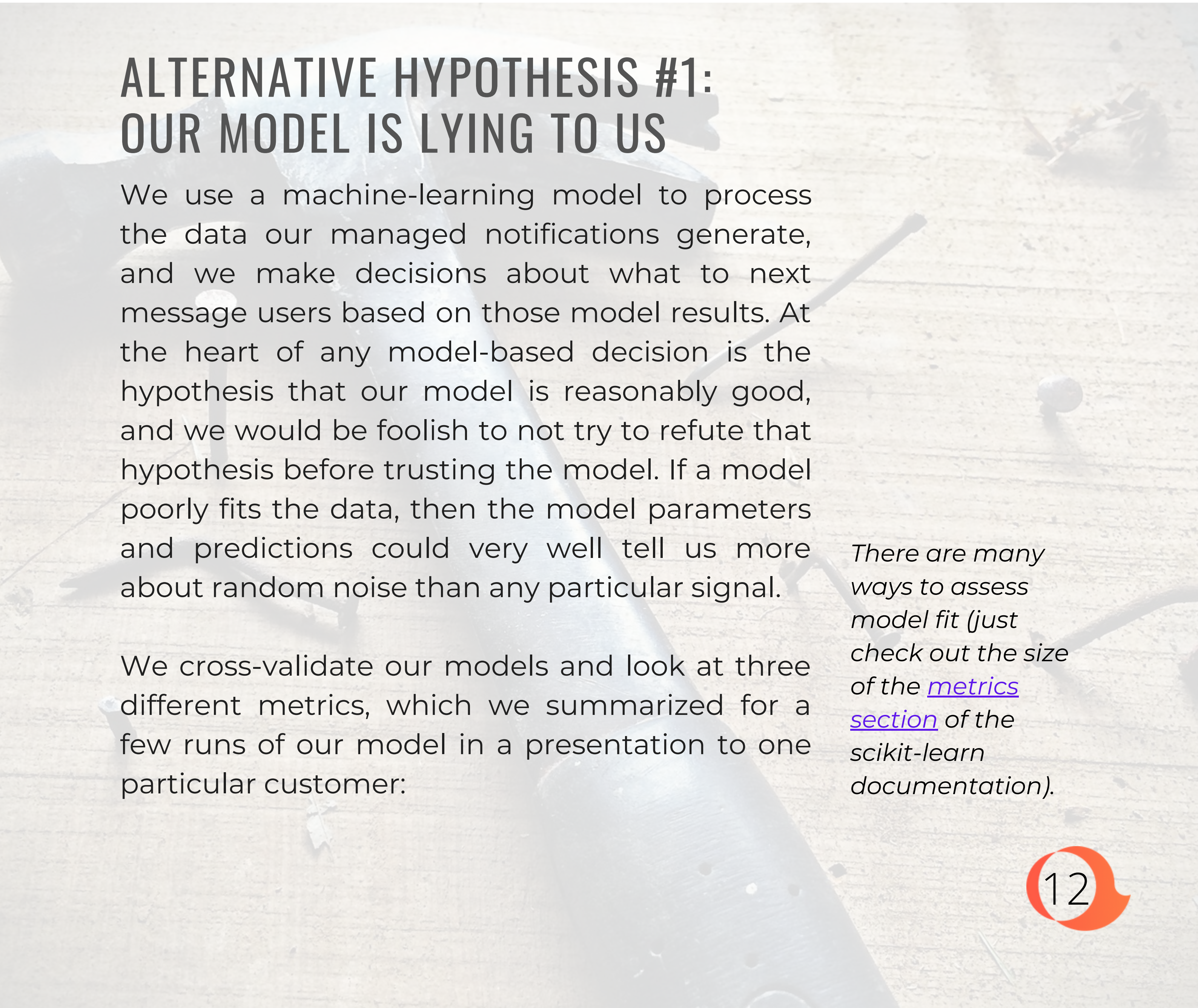
- We're taking credit for things users were going to do anyway - we predicted, but we didn't influence.

- Our messages prompted people to act, but the specifics of the message mattered much less than the simple fact that the person got the notification.

- We messaged the statistically right thing at the statistically right time for each user, modeling general trends rather than individual preferences.

- We're just improving general engagement but not influencing the things that really matter to our customers.

Let's walk through each of these alternative hypotheses and look at what we do to mitigate each one.

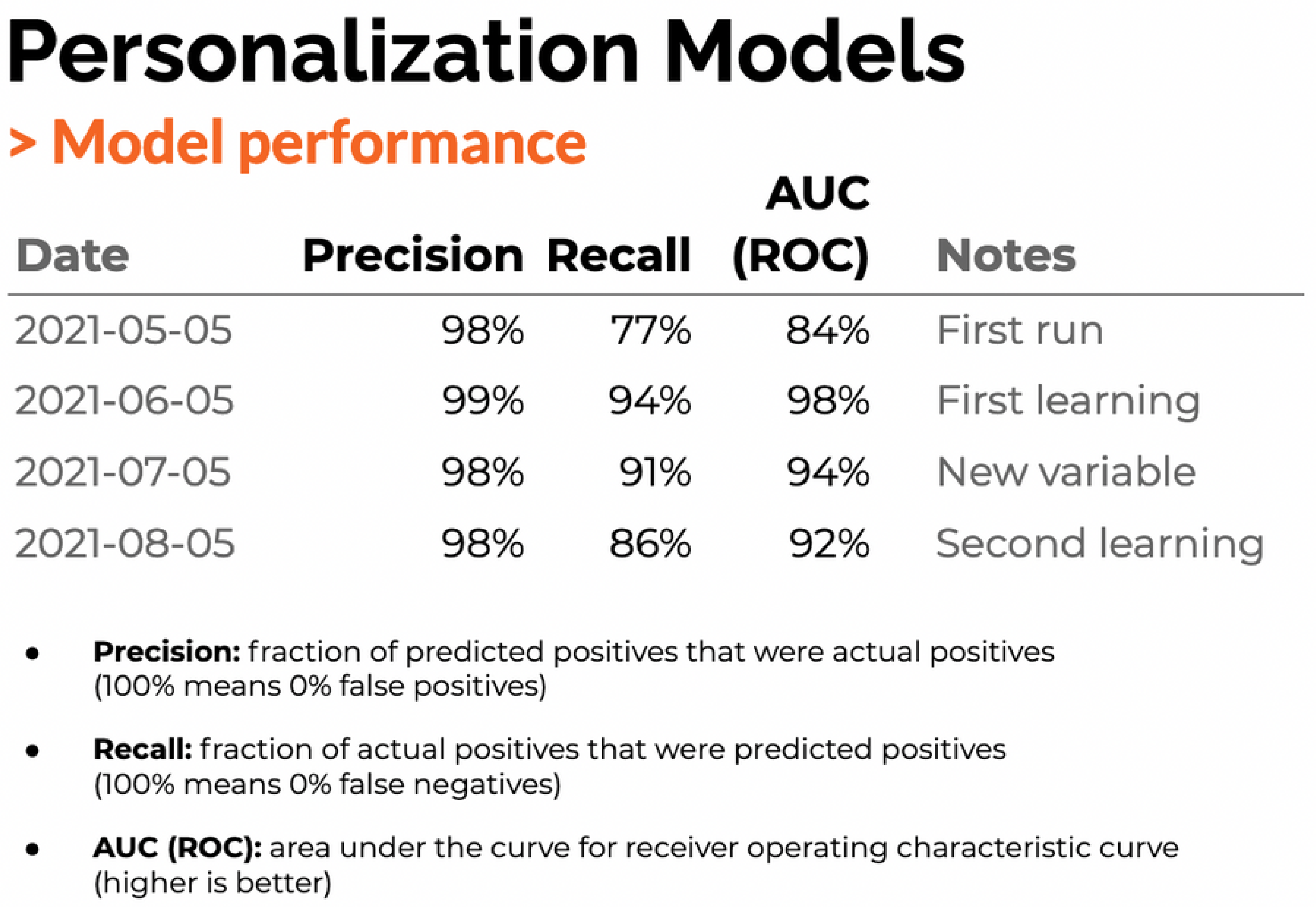


ALTERNATIVE HYPOTHESIS #1: OUR MODEL IS LYING TO US

We use a machine-learning model to process the data our managed notifications generate, and we make decisions about what to next message users based on those model results. At the heart of any model-based decision is the hypothesis that our model is reasonably good, and we would be foolish to not try to refute that hypothesis before trusting the model. If a model poorly fits the data, then the model parameters and predictions could very well tell us more about random noise than any particular signal.

We cross-validate our models and look at three different metrics, which we summarized for a few runs of our model in a presentation to one particular customer:

There are many ways to assess model fit (just check out the size of the [metrics section](#) of the scikit-learn documentation).



Personalization Models

> Model performance

Date	Precision	Recall	AUC (ROC)	Notes
2021-05-05	98%	77%	84%	First run
2021-06-05	99%	94%	98%	First learning
2021-07-05	98%	91%	94%	New variable
2021-08-05	98%	86%	92%	Second learning

- **Precision:** fraction of predicted positives that were actual positives (100% means 0% false positives)
- **Recall:** fraction of actual positives that were predicted positives (100% means 0% false negatives)
- **AUC (ROC):** area under the curve for receiver operating characteristic curve (higher is better)

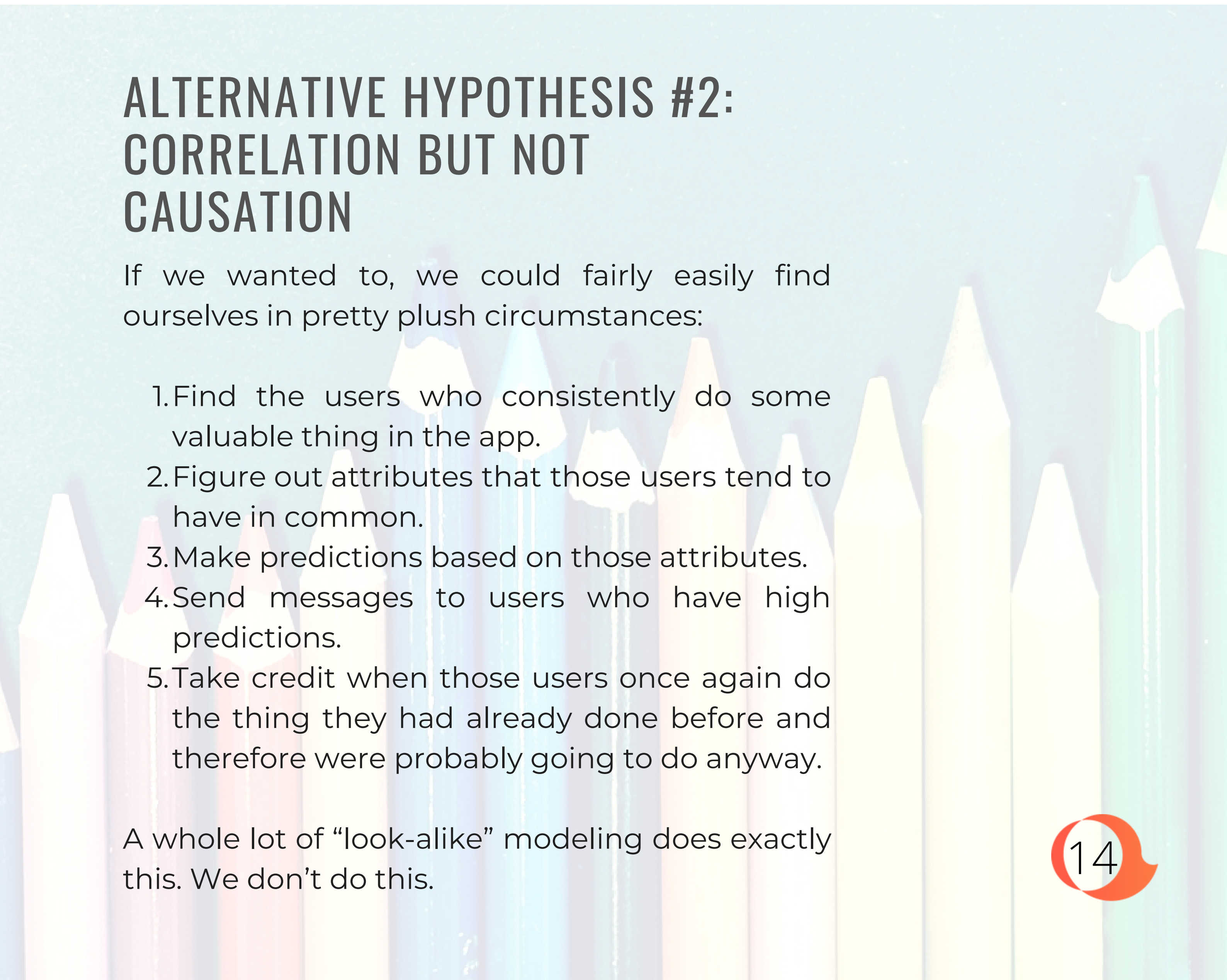
In all cases, all of these values are respectable, and in most cases they are quite good. A couple things to note:

- Precision and recall require a decision threshold. We always use a probability of 0.5 as the boundary between "the model predicts the user won't act" and "the model predicts the user will act". We use that threshold because we balance the classes in our model (so it learns from the same number of people who didn't do a thing as people who did), so we expect that a 50% probability really should indicate a 50% chance of doing a thing.

You can read more about precision and recall and receiver operating characteristic if those topics interest you. It's enough here to say that each is a percentage, and in each case a higher percentage is better.

- Notice that all three metrics drop during the third week in the table above and two of them drop again in the fourth week. Our model had spent two weeks learning based on one set of inputs, and in the third week it had to consider an additional variable that we threw into the mix. So we expected the model to have difficulty with new information.

The evaluation metrics we generate from our models consistently show that they make reliably high-quality predictions. So we have several good reasons to not believe that our system is just feeding us random noise.



ALTERNATIVE HYPOTHESIS #2: CORRELATION BUT NOT CAUSATION

If we wanted to, we could fairly easily find ourselves in pretty plush circumstances:

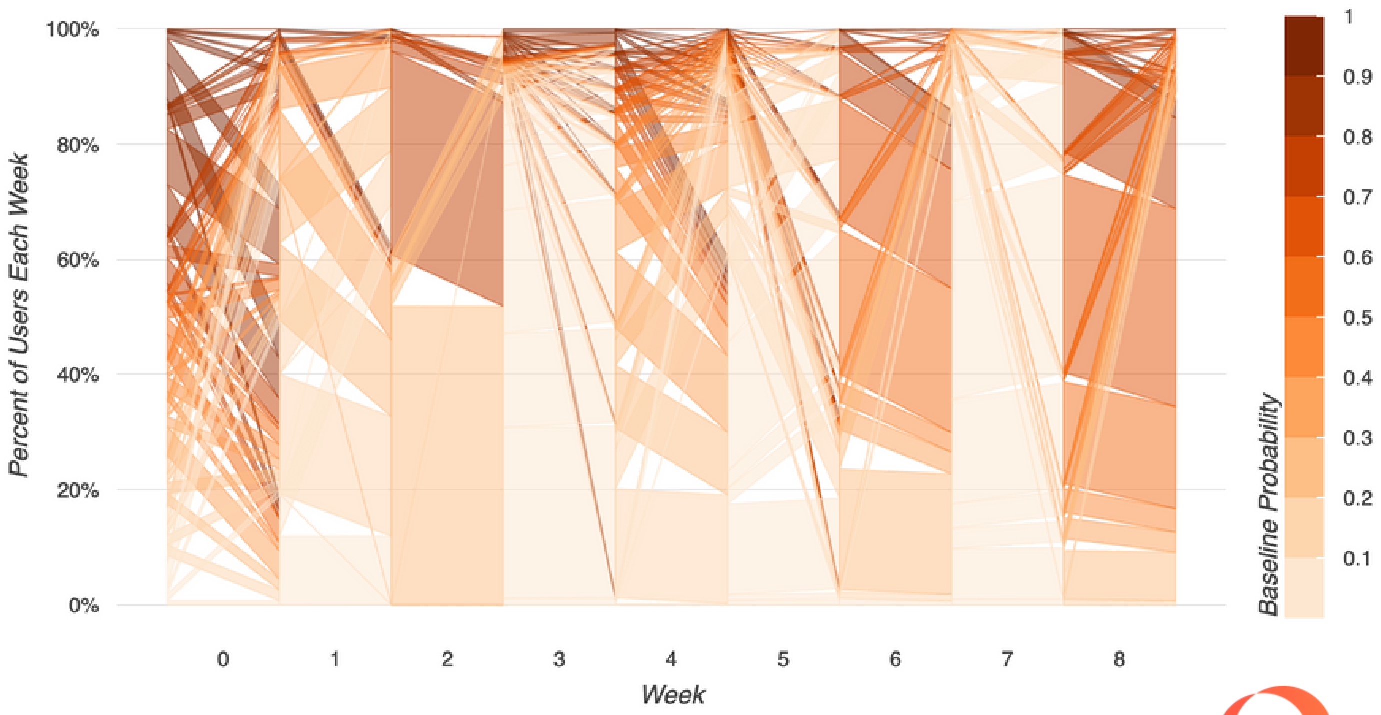
1. Find the users who consistently do some valuable thing in the app.
2. Figure out attributes that those users tend to have in common.
3. Make predictions based on those attributes.
4. Send messages to users who have high predictions.
5. Take credit when those users once again do the thing they had already done before and therefore were probably going to do anyway.

A whole lot of “look-alike” modeling does exactly this. We don’t do this.

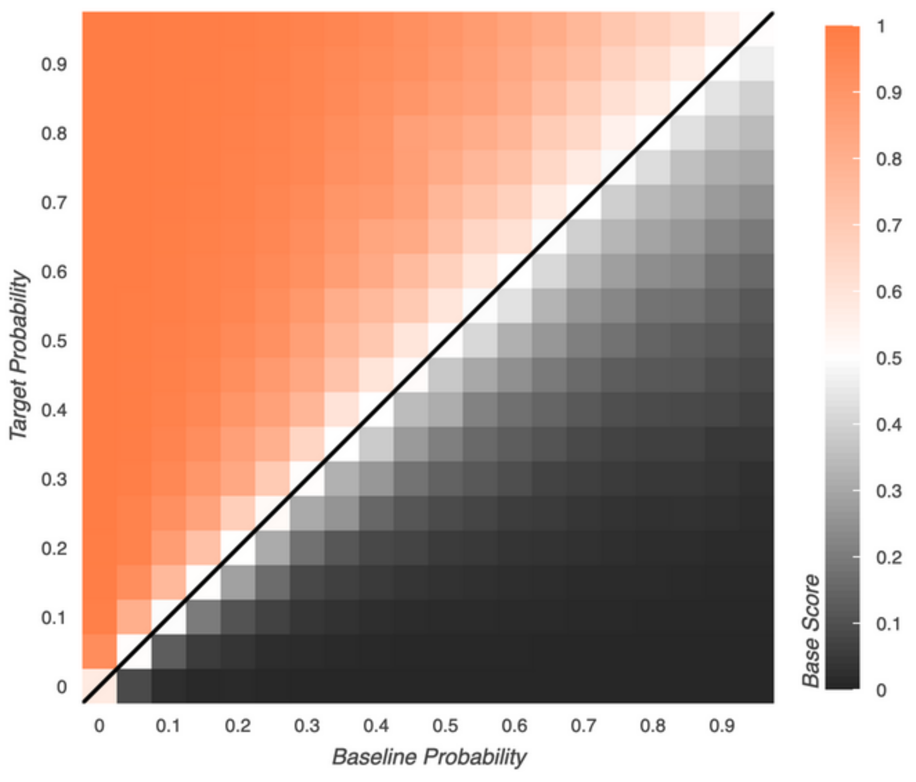
We don't use our model predictions directly in our decisions. Rather, we use what the model learned about associations between various baseline conditions (what we call The User Landscape) and messaging details (what we call assignments) on the one hand, and user behavior on the other hand. This is why we call it a *"what-if?"* model: the structure of the model allows us to ask "what if we'd messaged this user on a Tuesday instead of a Sunday?", but it also allows us to ask "what if we hadn't made any assignments at all?" This is what we call our *baseline probability*.

The baseline probability is the model's judgement about how likely a user would be to do the thing we wanted them to do, based solely on the attributes, past behavior, and all those other things that *don't* involve us putting a notification in front of them. This is what those baseline predictions looked like over several weeks for one of our customers:

The chart shows the percentage of users who move from one level of baseline probability to another over multiple weeks. The thicker the band, the more users, and the darker the band, the higher the initial baseline probability.



The baseline changes each week depending on what we were able to achieve, so sometimes a lot of users have a low baseline one week and a high baseline another. We then compare this baseline probability to a “target probability”, which is what we get when we ask our model “how likely would this user be to act given the user’s baseline data plus a particular message assignment?” The relationship between baseline probability and target probability determines a base score - called that because it forms the base upon which we build our personalization scores:



Think of this as a lookup table: find a user’s baseline probability along the horizontal axis and their target probability for a particular messaging decision on the vertical axis. The color and shade of the cell tells you what their base score will be: the darker the black, the lower the score; the brighter the orange the higher the score.

There are several important things to notice about this chart:

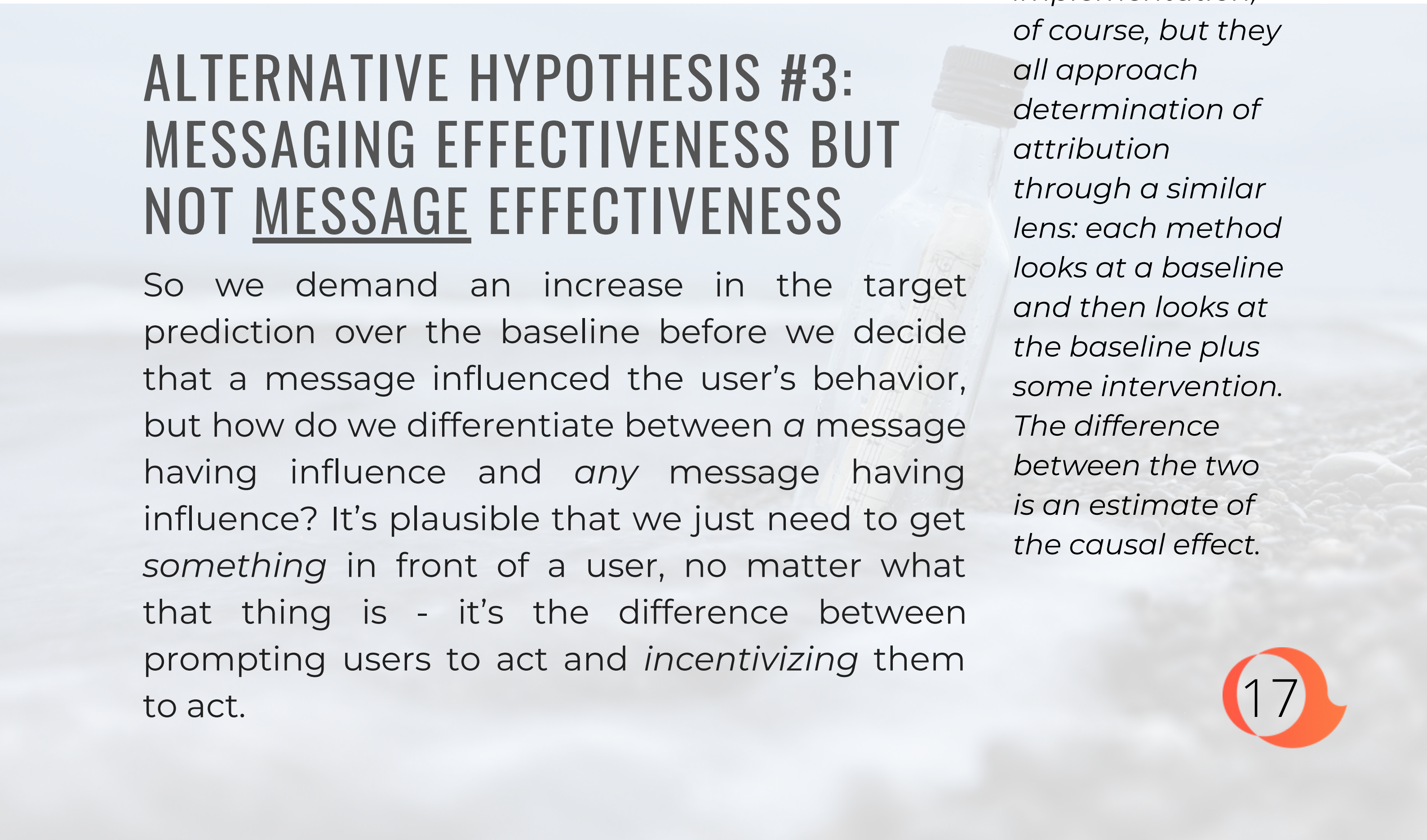
- Users whose baseline probability is equal to their target probability get a base score of 0.5.

- There are users with an extremely low target probability (less than 0.1) who still get a very high base score (greater than 0.9) because their baseline probability is so low.

- There are users with a pretty high target probability (greater than 0.7) who still get a pretty low base score (lower than 0.3) because their baseline probability is already high.

So our base score doesn't answer the question "how likely is this user to act?". Rather, it answers the question "how likely is this user to act if we make a particular decision about their notification?" It can be a good decision to message someone who is unlikely to act in general but is five times more likely to act if we message them. It can be a bad decision to message someone who is likely to act even if we don't message them at all.

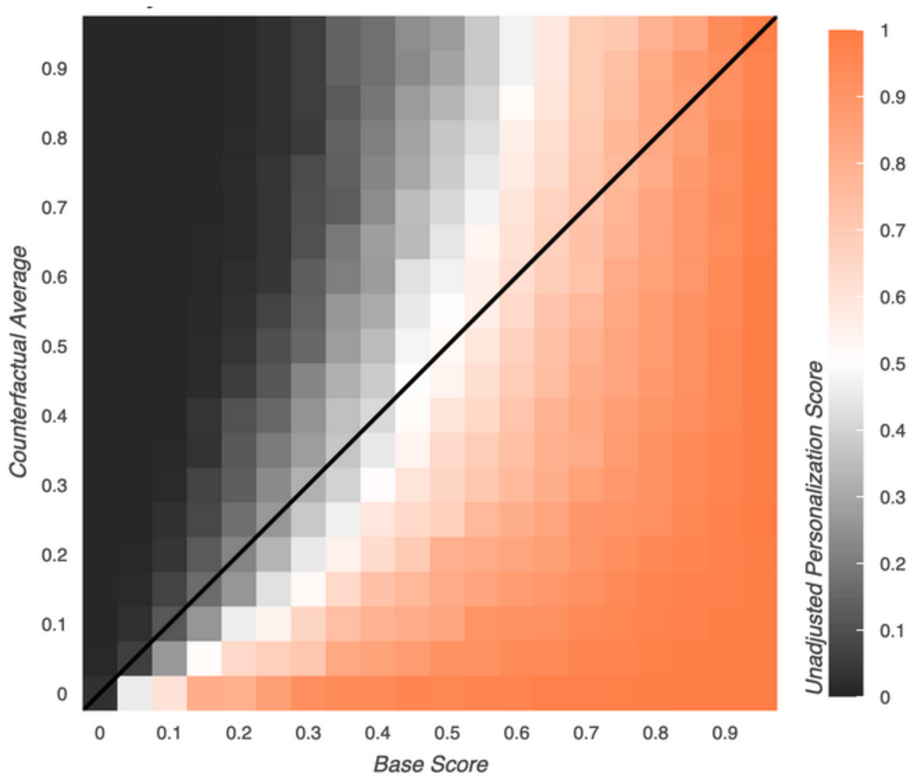
Incidentally, this method is conceptually similar to methods like Granger causality and vector autoregression used in econometrics. All three methods are quite different in implementation, of course, but they all approach determination of attribution through a similar lens: each method looks at a baseline and then looks at the baseline plus some intervention. The difference between the two is an estimate of the causal effect.



ALTERNATIVE HYPOTHESIS #3: MESSAGING EFFECTIVENESS BUT NOT MESSAGE EFFECTIVENESS

So we demand an increase in the target prediction over the baseline before we decide that a message influenced the user's behavior, but how do we differentiate between *a* message having influence and *any* message having influence? It's plausible that we just need to get *something* in front of a user, no matter what that thing is - it's the difference between prompting users to act and *incentivizing* them to act.

We calculate a "counterfactual average" for each of our base scores. If we're learning what day of the week to send notifications, the counterfactual average for Monday is the average of Tuesday's through Sunday's base scores. The higher the base score is relative to the average, the more evidence we have that the important action was sending on a Monday and not just sending a message in general. This results in an "unadjusted" personalization score:



This looks similar to the previous graph, showing the calculation of the base score, but notice that the white spaces (indicating a result near 0.5, indicating the realm of "we don't know how a user feels about this") don't follow the black diagonal line.

If a user has a counterfactual average more-or-less equal to that user's base score, then the base score remains unchanged. If the counterfactual average is lower than the base score, then we inflate the base score because that means, no matter how poorly a user may respond to that assignment, it's better than the alternatives. If the counterfactual average is higher than the base score, then we deflate the base score because, no matter how well a user may respond to that assignment, there are better choices available.

ALTERNATIVE HYPOTHESIS #4: SEGMENTATION BUT NOT PERSONALIZATION

Basing decisions on modeled probabilities runs the risk of smoothing over individually-important differences. That's because models are only as good as the data we feed into them. Remember: none of our model validation scores were at 100%. That means there are other factors - perhaps just random noise but perhaps something more systemic - that influence outcomes, and our models aren't capturing that. So it's entire possible for a user to:

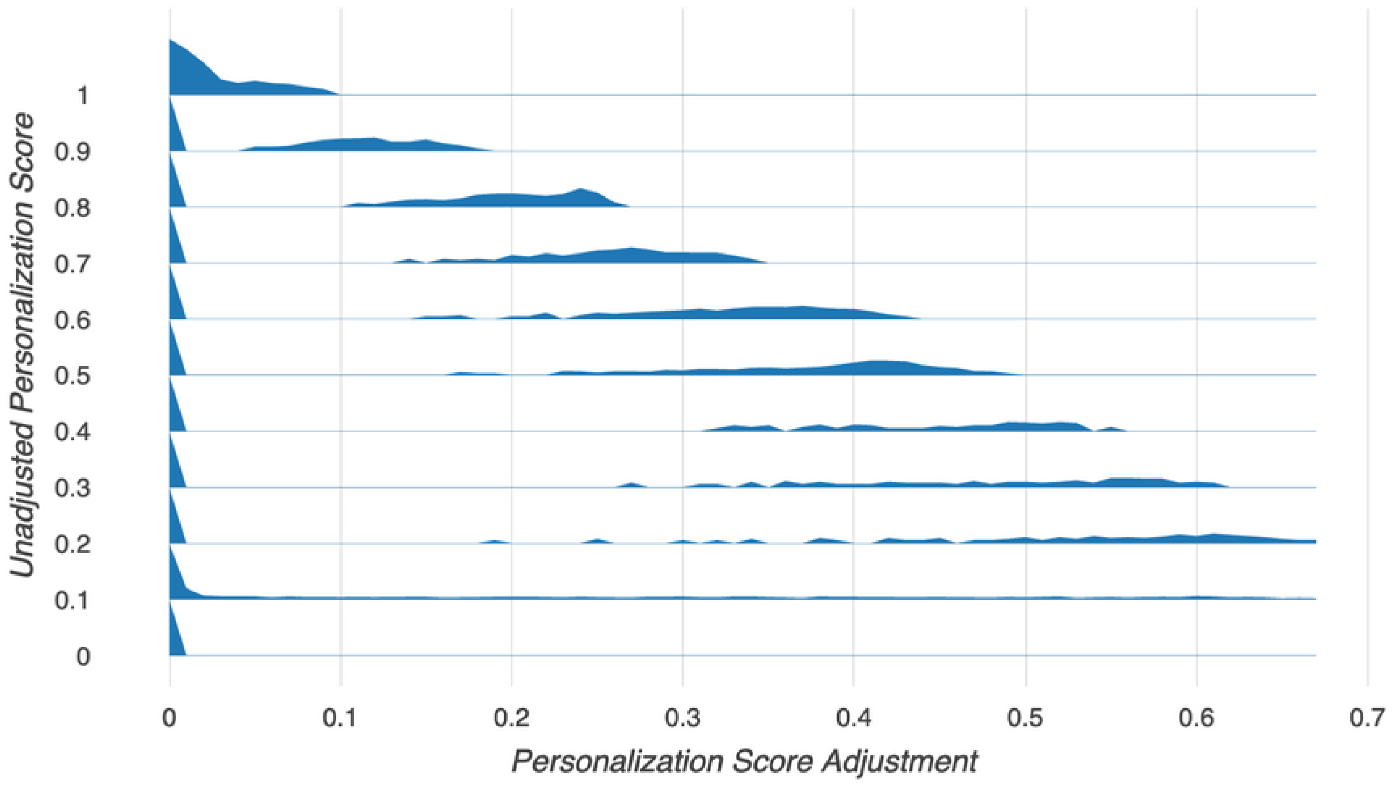
1. Do the thing we want them to do;
2. Be very similar to many other users who didn't do the thing; and
3. Get a message assignment that doesn't tend to work out as well for other users.

Under those conditions, the user is going to get a low personalization score. This is what segmentations tend to do - find what works for a lot of people on average and then act as if the preferences of every individual in the segment fit that average. That works ok in some situations, but not in all situations, and besides, we can do better.

If a user reacts to our message, we inflate the personalization score. If the reaction is unrelated to our customers' goals - just a brief visit the app, but nothing more - we boost their score a little. If the reaction is the exact thing we had as a goal, we boost it more.

We can also boost base on when a user reacts: if they act two days after we message, we boost them a little; if 30 seconds after we message, we boost them a lot.

This chart shows how that adjustment has tended to work out recently for one of our customers:



Each row of the chart represents a bin of unadjusted personalization scores: scores between 0.0 and 0.1, between 0.1 and 0.2, and so forth. The shaded area on each row shows the distribution of upward adjustments we made based on user activity.

You can easily see a couple things:

- Users who had higher unadjusted scores tended to get less of a boost, because they were already high and therefore had less movement available before they hit the maximum score of 1.0.
- Most users got no boost at all - the shaded areas are thickest at the left side of the chart, indicating a boost of zero or near-zero. But in cases where users did react, their scores reflect that.

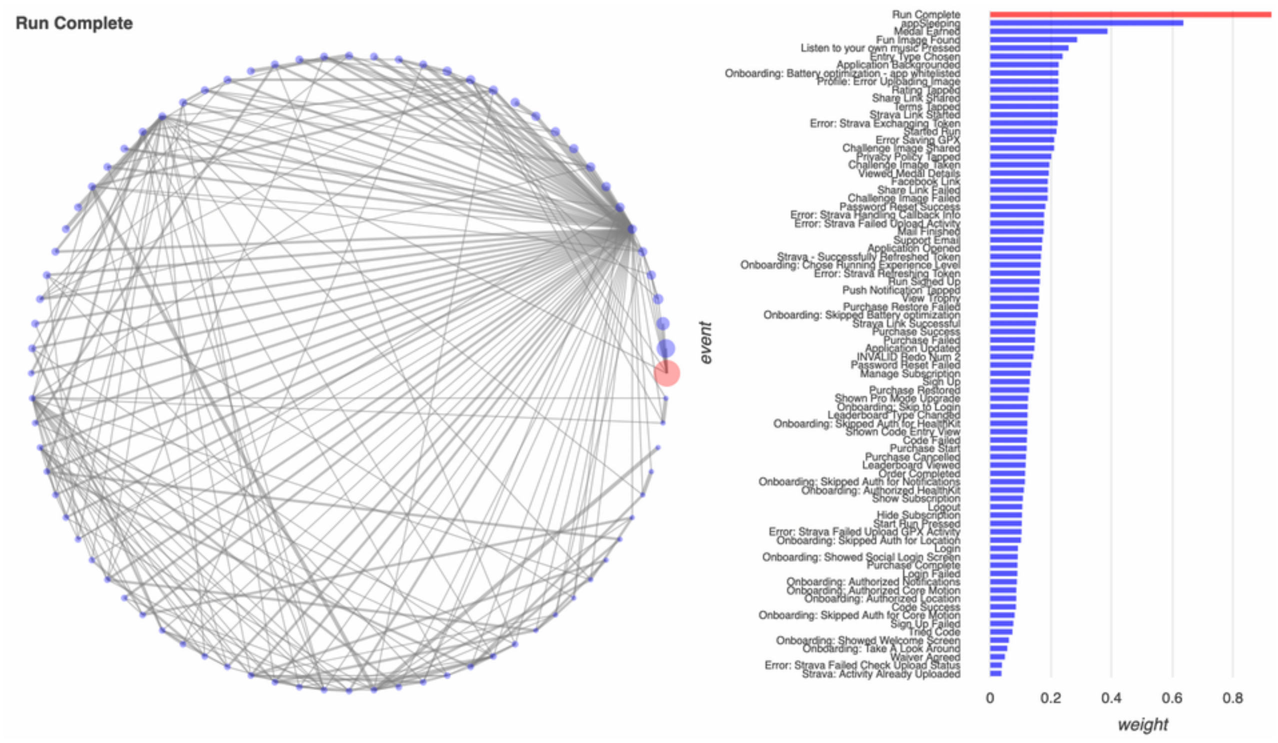
ALTERNATIVE HYPOTHESIS #5: GENERAL ENGAGEMENT BUT NOT TARGETED GOAL ACHIEVEMENT

One criticism against claims of digital advertising and marketing effectiveness comes up so often that it's kind of recognized as just how business messaging works: advertising drives clicks. Clicks aren't what most customers care about. Because we integrate with an app's event stream, we don't focus on clicks. If you want a user to actually watch a video, or actually make a purchase, or actually play a game, or actually do any of the things your app allows users to do, we can measure that directly, and we can focus our learning systems on those goals.

People don't obsess over click-through rates because they can easily see how a click leads to value for their company. They obsess because they know that not clicking leads to no value. The click isn't what they care about, but it's the only thing they can measure, so they focus on clicks and try not to think too much about the dropoff that happens between a click and the achievement of any meaningful goal.

In fact, we go even further than that. Every event feeds into other events. Some events lead consistently and immediately to another, while other events lead to many other events, maybe after quite some time, depending on the nature of the event and the specific detail of what the user wants. These are user journeys. When our customers identify a goal event, we use all of the information we've collected about user journeys to create weights that we then use to train our models.

This is the Journey Graph we mentioned earlier:



Naturally, the actual goal event gets the highest weight, but any events that consistently feed into that goal can receive substantial weight as well. This not only gives our models more data to work with, but also makes our personalization scores reflect the probability of actually doing the thing our customers care about most, which in most cases, isn't just general engagement.

SO HOW DO WE KNOW IT WORKS?

The previous section shows that we work to mitigate considerations that could undermine our preferred hypothesis by plausibly rendering us unable to reject any of the alternatives. It doesn't show how well any of those mitigating strategies work.

THE CONCEPT OF LIFT

Lift is fundamentally a metric of attribution, and attribution, particularly when it comes to human behavior, is never straightforward. Any user has many different needs, wants, preferences, and constraints influencing their behavior. We have to navigate those preferences and sometimes arrange those constraints to get them over the line from "I could do that thing" to "I'm going to do that thing" and eventually to "I did that thing."

A lift calculation involves counting up how many users you influenced, which means it necessarily depends upon the story you tell to justify taking credit for an outcome - and that's why it's a powerful way to assess competing hypotheses. Hypotheses are themselves stories, so the method fits the objective. Unlike goodness of fit or statistical significance, lift estimates practical, real-world impact. For any competing hypothesis, what would that impact metric need to look like for the hypothesis to be true? Answering that question gives us a basis for challenging those hypotheses.

If you kick a ball, you cause that ball to move - it wouldn't have moved in that moment if you had not applied the force of your foot to it. You don't cause users to do things on an app the same way you cause a ball to move.

We make decisions about what and how to message based on users' adjusted personalization scores, which uses all of the safeguards and mitigating strategies discussed in the previous section. We go through a pretty involved process of combinatorial optimization to make sure we balance the desire to give users their preferences with the need to keep learning about those preferences.

We don't just pick the best score for each user: e.g. "we have to decide what day of the week to message them and their highest score is for Thursday so we'll send them a message on Thursday".

If we just picked everyone's top-scoring item as the winner, it's very possible that a bunch of users who were all very similar (in terms of their attributes and past behavior) would get the same assignment. That would deprive us of the ability to continue learning about that group's preferences (because we can only learn preferences when we have multiple options to compare). So we take groups of similar users, and figure out when someone's second or third preference is still good enough that by giving it to them, we can give other people their first preferences, and therefore still maintain a situation where we have multiple preferences at play.

All that being said, there is always, necessarily, a pretty strong relationship between the magnitude of a user's personalization score and the messaging choice we assign to them. Their highest score might be quite low in absolute terms - we do see users who show no marked preference for any day of the week or any time of day - but their assignment will tend to correspond to one of the highest scores they personally happen to have.

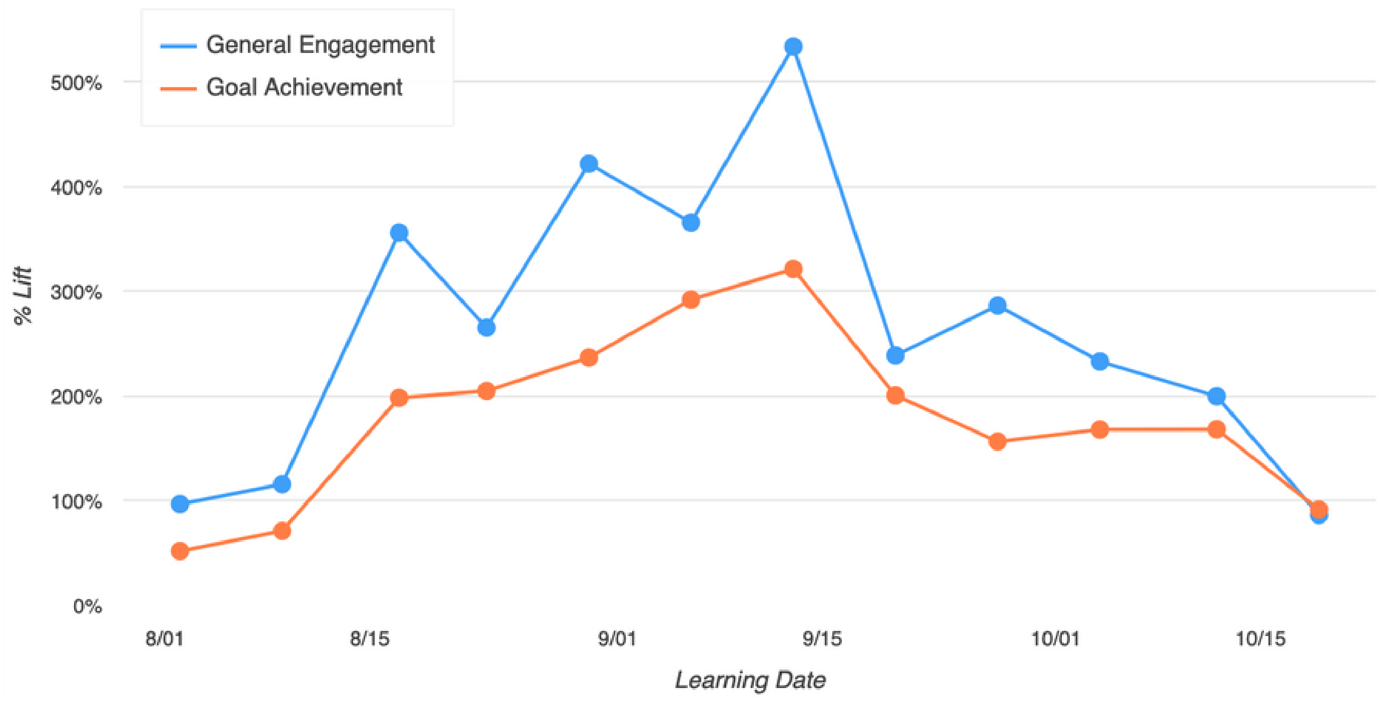
This relationship between previous score and subsequent assignment gives us a basis for calculating lift:

1. Divide messaged users into two groups: those who did an action (meaning they triggered a particular app event) after they were messaged, and those who didn't. Deciding how long to monitor a user's activity after a message is a tricky issue. We monitor for several days, but weight activity that happens within minutes or hours of the message much more highly.
2. For each group, compose an empirical distribution representing the personalization scores used to make a particular messaging decision. So if we're looking at lift from timing choices, we'll look at the personalization score we used to decide to send a particular user a notification at, say, Monday at 9:00am rather than Wednesday at 3:00pm.
3. Estimate the percent of the distribution of those who did the action that exceeds the distribution of those who didn't do the action. This is essentially the probability that a random draw from the did-the-thing distribution is higher than a random draw from the didn't-do-the-thing distribution. This is the probability that successful messages were based on better-quality decisions than unsuccessful messages.
4. We can convert that probability to an odds ratio - the number of successes based on good decisions relative to the number of successes based on bad decisions. Subtract one from that, and we get the percent increase in successes that resulted from good decisions. That's the lift.

The more a user does something directly related to our customer's goals, and the sooner they do it relative to when we message them, the higher their weight is.

If personalization score makes no difference at all in getting users to act, then the two measures will be roughly the same, so the lift will hover right around 0.0%. Essentially, instead of picking specific users and saying “we influenced these”, we compare those who did the thing we wanted them to do to those who didn't do the thing, and estimate the extent to which our personalization scores explain the difference between the two groups. This is what that looked like for a few weeks for one of our customers:

We would expect 0% lift if we randomly assigned personalization scores.



A couple things to note:

- The lift was always positive, even for the first week shown, which was only one week after we started messaging users and reflected the first week we tried to learn about these users.

- Goal achievement lift was always lower than general engagement lift, which isn't surprising since it's a higher bar to meet.

The two kinds of lift don't move in lock-step. Improving general app engagement doesn't necessarily result in improving the specific goals the customer laid out, and improving goal achievement doesn't necessarily correspond to better general engagement.

Keep in mind that these lift numbers are based on personalization scores that we used to make assignments before we knew how the user was going to act. We learned during week 0, calculated personalization scores and made an assignment based on that score at the beginning of week 1, observed what users actually did at the end of week 1, and then calculated the lift at the beginning of week 2. So there isn't a way for us to stack this deck - we didn't know what was going to happen at the time we used the personalization scores to make assignments.

USING LIFT TO TEST HYPOTHESES

We can use our lift metric to decide how well we're doing at refuting or mitigating all of those alternative hypotheses we listed earlier:

- **Random score**. If lift based on randomly assigned scores is greater than or equal to the lift based on our personalization scores, then our model doesn't actually work.

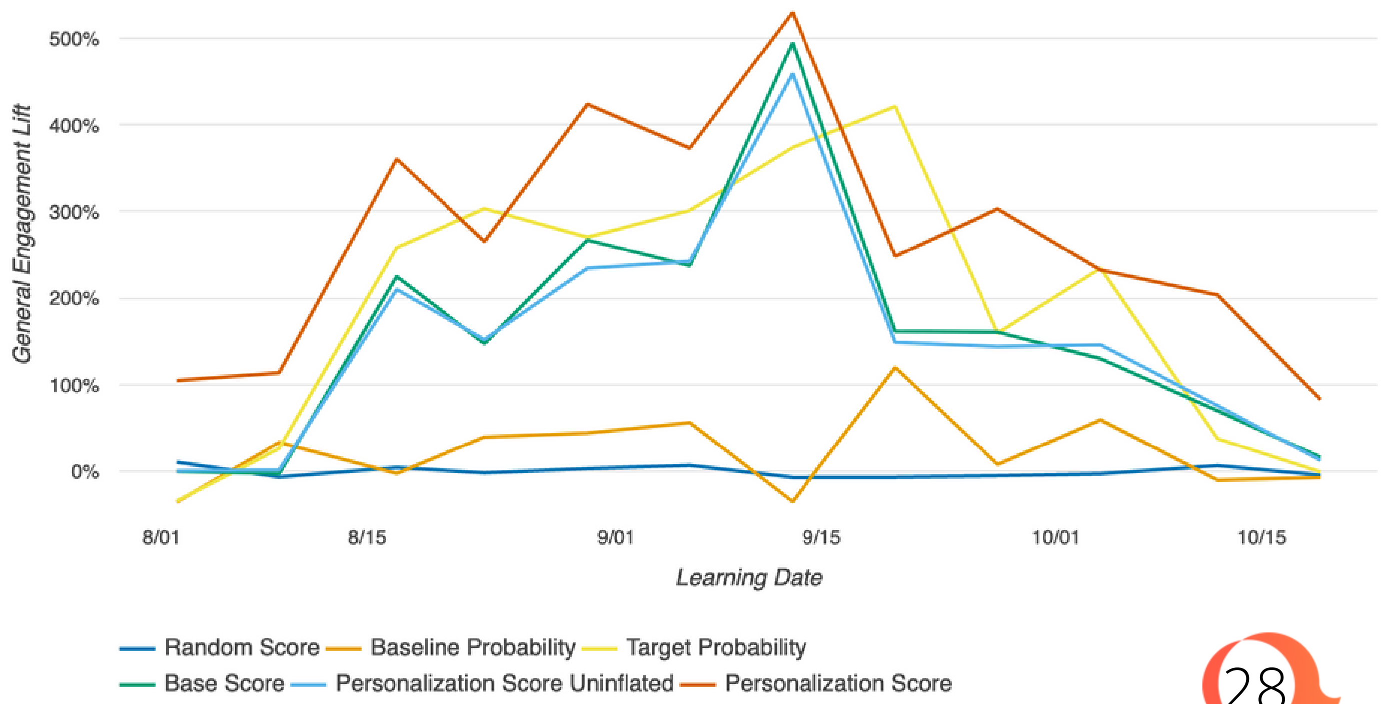
- **Baseline probability**. If lift based on the baseline predictions from our model is greater than or equal to lift based on our personalization scores, then we just predicted which users were likely to act, rather than actually influencing them.

- **Target probability and base score.** If our lift based on target probability is greater than or equal to the lift based on base score, then the fact that we messaged people matters more than the specific details of the notification.

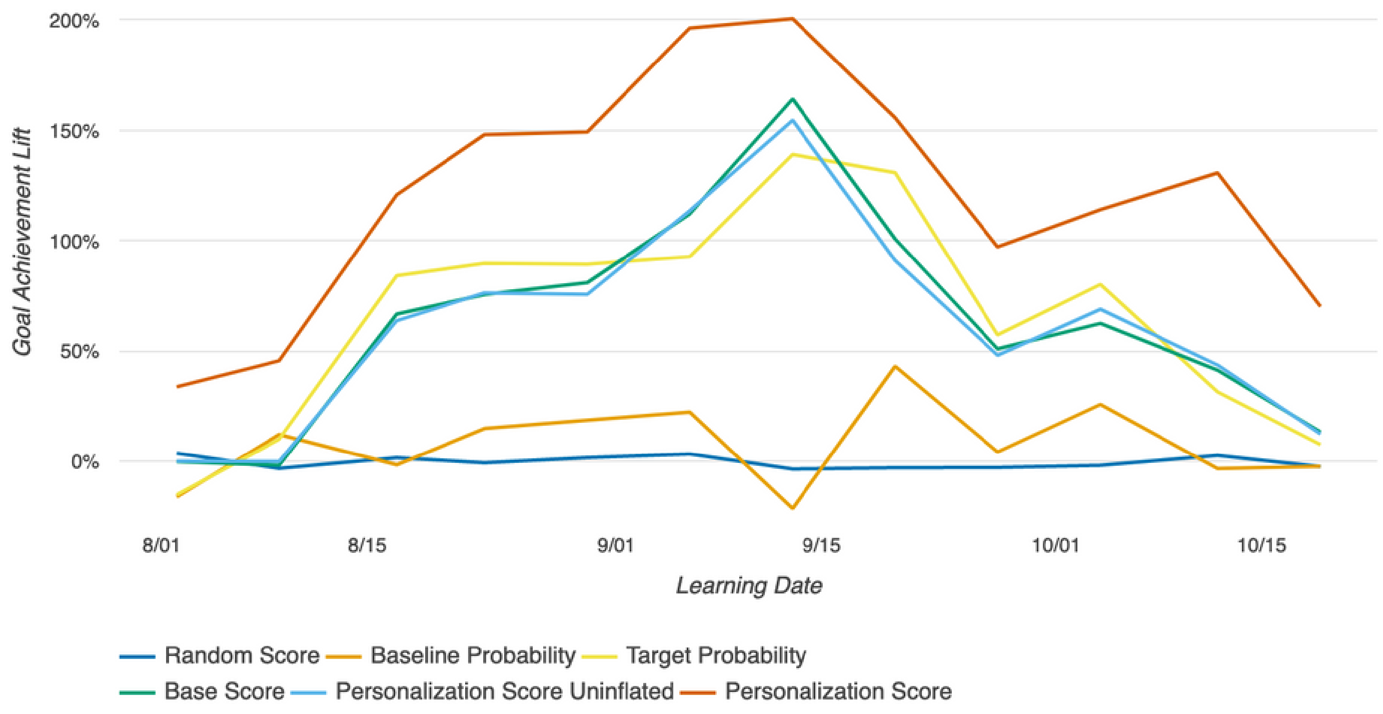
- **Uninflated personalization score.** If the uninflated personalization score is greater than or equal to the lift based on personalization scores, then we're essentially segmenting instead of personalizing.

- **Targeted engagement.** If our general engagement lift outpaces our targeted engagement lift relative to the other metrics I've listed here, then that means we're influencing general engagement but not necessarily honoring our customer's priorities.

Here's how all that works out for general engagement lift:



And here's how it looks for targeted engagement lift:



Several things to notice:

- The lift based on random scores is consistently around 0%, which is what we expected, which means our lift metric works the way it is supposed to - it's reasonably well-calibrated. This metric is also quite a bit less than the lift based on personalization scores, so it's a strong refutation of alternative hypothesis #1.

- Lift based on baseline probability is not only much less than lift based on personalization score - it's just a little bit higher than lift based on random noise. So this is a strong refutation of alternative hypothesis #2.

- Target probability is usually greater than base score for general engagement lift, and is often above base score for targeted engagement lift. That means we can't refute alternative hypothesis #3 - that it's the fact of messaging that matters more than the details of this message.

- The uninflated personalization score is consistently lower than the full individualized personalization score, so it's at least a moderate refutation of hypothesis #4.

- The differences between lift based on personalization scores and lift based on other metrics is more consistent and pronounced for targeted engagement than it is for general engagement. That's at least moderate refutation of alternative hypothesis #5.

The learning that produced the data here focused solely on message timing - we weren't systematically managing message content for this customer at this particular point - so this result is not surprising.

Those are strong results. We'd like to see more definitive refutation of alternative hypotheses #4 and #5, and we don't have the data yet to reach a conclusion about alternative hypothesis #3 (we always have our customers start by optimizing timing because the only way to tell if your message was right is if you're reasonably sure you at least sent it at the right time). We were frankly a little surprised that our general engagement lift looked so good, given that that wasn't ever our focus.

After two weeks of learning, we're able to provide our customers with lift that tends to hover in the range of 100% to 200%. That's powerful.

WE ONLY NEED TO BE RIGHT ENOUGH.

One last point about all of our learning systems. People often seem to think of results of a test as things that you have to be prepared to live with for a long time. So you have two ideas for your landing page, you implement both, run an A/B test, and pick the winner and take the loser down.

Maybe this is because people are most familiar with the A/B testing paradigm (and the analogous null-hypothesis significance testing paradigm in statistics).

That is living so very far below your means.

Information about bandit algorithms has not penetrated into business-world consciousness nearly as far as A/B tests have, and contextual bandits have made even less headway. These alternatives operate on the very simple principle that you should hedge bets instead of picking winners. The only reason it makes sense to run a test and simply live with the result going forward is if you can't afford to do it in a better way.

How to hedge a bet: if A works twice as well as B, show A twice as often but keep showing B, because there may be a relatively small subset of users who really prefer B, or because A might be the preference right now but B will be the preference next month.

We don't need our model to be right forever. In most cases, we need it to be right for a week - at that point, we'll have sent more notifications, learned new preferences, and updated our personalization scores. The "right" answer today will almost certainly be wrong later, and the "wrong" answer today is probably right for some users. There's no reason to ignore those challenges. With a system that can keep learning, automatically, without you needing to make manual decisions about every little result, you can adapt as fast as your users do.

aampe