Evaluating Mobile Apps with A/B and Quasi A/B Tests

Ya Xu
LinkedIn Corporation
2029 Stierlin Court
Mountain View, CA, 94043
yaxu@linkedin.com

Nanyu Chen
LinkedIn Corporation
2029 Stierlin Court
Mountain View, CA, 94043
nchen@linkedin.com

ABSTRACT
We have seen an explosive growth of mobile usage, particularly on mobile apps. It is more important than ever to be able to properly evaluate mobile app release. A/B testing is a standard framework to evaluate new ideas. We have seen much of its applications in the online world across the industry [9,10,12]. Running A/B tests on mobile apps turns out to be quite different, and much of it is attributed to the fact that we cannot ship code easily to mobile apps other than going through a lengthy build, review and release process. Mobile infrastructure and user behavior differences also contribute to how A/B tests are conducted differently on mobile apps, which will be discussed in details in this paper. In addition to measuring features individually in the new app version through *randomized* A/B tests, we have a unique opportunity to evaluate the mobile app as a whole using the *quasi-experimental* framework [21]. Not all features can be A/B tested due to infrastructure changes and wholistic product redesign. We propose and establish quasi-experimental techniques for measuring impact from mobile app release, with results shared from a recent major app launch at LinkedIn.

Keywords
A/B testing, mobile, quasi-experiments, causal inference

1. INTRODUCTION
It is clear that mobile is taking over the Internet. In the U.S., two out of three digital media minutes is now happening on mobile. Without question, mobile app usage has been the single most important driver by far, contributing to nearly 90% of the massive mobile growth in the past two years [5]. At LinkedIn, more than 50% our traffic is now coming from mobile, similar to other social networking companies such as Facebook and Pinterest [3].

Many companies, including LinkedIn, have shifted to a "mobile first" mindset, focusing their product strategies around mobile. As a consequence, optimizing for the best user experience in mobile has been more important than ever. However, mobile optimization is a space that is a lot less mature than web optimization. Because of how users consume information on mobile differently, many lessons learnt by optimizing on the web no longer apply. So even companies most experienced with web optimization have to start from the bottom and relearn in the mobile space.

It is not just about coming up with "what" works in mobile, but more importantly, we need to have methodology and system in place to "measure" whether an idea works. A/B testing, also known as controlled experiment, is a standard, widely used framework to evaluate new ideas and to make data driven decisions. We have seen much of its applications in the online world discussed in recent publications, including several past KDD papers from Google, Microsoft and LinkedIn [9,10,12]. However, none of these papers have a focus on mobile, or mobile apps.

KDD '16, August 13-17, 2016, San Francisco, CA, USA
© 2016 ACM. ISBN 978-1-4503-4232-2/16/08 $15.00
DOI: http://dx.doi.org/10.1145/2939672.2939703

At LinkedIn, we have seen a drastic increase of mobile experiments. The growth is more than just proportional to the amount of development work relative to desktop. We have learnt that because the real estate on mobile is limited, small changes tend to have big impact. In addition, A/B testing on mobile is used even more extensively than it is on desktop, and much of it is attributed to a key difference between desktop and mobile development process. Because we cannot ship code to the mobile app other than building and releasing a new app version, feature releases in general involve not just the app developers, but also the app stores (e.g. Google Play or Apple App Store), and the end users. The app store usually requires a review of the build submitted, and a new app version is not effective until the end users update the app. The consequence is that it may take two weeks or more for any change to reach the majority of users, and even longer for a coverage of greater than 90%. A/B testing is hence heavily leveraged to mitigate risk, as it allows us to evaluate the new feature and gradually release it accordingly, without having to release a new version.

The fact that we cannot ship code easily to mobile apps, together with mobile infrastructure and user behavior differences, strongly influences how A/B tests are conducted on mobile apps. We will discuss them in depth in Section 4, following the three steps in the A/B testing process: design, deployment and offline analysis.

In addition to measuring features individually in the new app version through *randomized* A/B tests, we also have a unique opportunity to evaluate the mobile app as a whole using the *quasi-experimental* framework [21]. Not all features can be A/B tested due to infrastructure changes and limitations. For example, LinkedIn recently rewrote the entire flagship app (Project Voyager [3]). Even though some new features were tested and evaluated independently in the old app, many features were launched for the very first time with the new app release. How all the features work together was a key question, but unfortunately we were not able to conduct a randomized experiment on the new app as a whole. However, because not all users adopt the new version at the same time, there is a period of time where we have both versions of the app serving real users. Obviously, simply comparing the adopters with the non-adopters will suffer from self-selection bias. However, we can still reasonably establish causal relationship by carefully removing such bias using quasi-experimental techniques (also called observational causal inference methods in some disciplines).

To set the notation and context for both A/B and quasi A/B testing, we review the Rubin Causal Model [23], a widely used framework to estimate causal effect. Let Y_i be the outcome

variable for user i, e.g. pageviews, clicks. Let $Z_i \in \{0,1\}$ be the treatment variable, where $Z_i = 1$ if a user is in treatment (adopted the new app), and $Z_i = 0$ otherwise. Note that in the case of mobile adoption, Z_i is *observed* rather than randomly generated. We also observe $\boldsymbol{X_i}$, a vector of pre-exposure covariates, such as industry or connection counts. In summary, we observe $(Y_i, Z_i, \boldsymbol{X_i})$ $for\ i = 1, \dots, n$. Under the Rubin Causal framework, each user has two potential outcomes

$$Y_i = \begin{cases} Y_{i1} & if \quad Z_i = 1 \\ Y_{i0} & if \quad Z_i = 0 \end{cases}$$

We are interested in knowing the Average Treatment Effect (ATE), the difference of the average outcomes between applying treatment to the *entire* user population and applying control to the *entire* user population. By definition,

$$\Delta_{ATE} = \frac{1}{n}\sum_{1}^{n} Y_{i1} - \frac{1}{n}\sum_{1}^{n} Y_{i0}$$

Of course, Δ_{ATE} is never known because only one of Y_{i1} and Y_{i0} can be observed (The fundamental problem of causal inference [22]). Instead, it is usually estimated by

$$\widehat{\Delta}_{ATE} = \frac{1}{n_1}\sum_{\{i,Z_i=1\}} Y_{i1} - \frac{1}{n_0}\sum_{\{i,Z_i=0\}} Y_{i0} = \bar{Y}^1 - \bar{Y}^0$$

Two important assumptions are required for $\widehat{\Delta}_{ATE}$ to be an unbiased estimator of Δ_{ATE}. (1) Stable Unit Treatment Value Assumption *(SUTVA),* which states that the behavior of each user in the experiment depends only on his own treatment and not on the treatments of others. (2) Ignorable Treatment Assignment Assumption (also known as *unconfoundedness*) [21], which states that treatment assignment is independent of the two potential outcomes, i.e.

$$(Y_{i0}, Y_{i1}) \perp Z_i$$

Both assumptions are naturally satisfied in most controlled experiments. However, the unconfoundedness assumption is often violated when there exists variables that correlate with both (Y_{i0}, Y_{i1}) and Z_i. As an example, we have observed that LinkedIn members with premium accounts are more likely to adopt a new version faster. Since premium members are in general more engaged on the site, we are likely to conclude a significantly positive impact even if the true Δ_{ATE} is 0 or negative. In fact, the unconfoundedness assumption can be extended to

$$(Y_{i0}, Y_{i1}) \perp Z_i \mid \boldsymbol{X_i}$$

where $\boldsymbol{X_i}$ are the confounding covariates. As a result, if we can identify all such covariates, we can still postulate a model for $E(Y_i|Z_i, X_i)$ and estimate ATE based on

$$\Delta'_{ATE} = \frac{1}{n}\sum_{i=1}^{n}(E(Y_i \mid Z_i = 1, X_i) - E(Y_i \mid Z_i = 0, X_i))$$

This is a fundamental concept of quasi-experimental framework. We will discuss more formally on this in Section 5.

Here is a summary of our contributions in this paper:

- As far as we know, we are the first to study extensively how to evaluate mobile app through both A/B testing and quasi A/B testing. Moreover, we are the first to propose and establish a quasi A/B testing framework to evaluate mobile app release.
- We identify the key differences of conducting A/B tests on mobile vs. on desktop.
- We compare and propose quasi-experimental techniques that have shown to be very successful in removing adoption bias, and are used in major mobile app release at LinkedIn.
- We share insights on how users adopt new app versions, including insights on how to measure novelty effect.

The paper is organized as follows. Section 2 starts with a review of the existing literature in all the three areas of A/B testing, mobile A/B testing and quasi A/B testing. Section 3 describes the process for mobile app releases and its implications on how we evaluate a new release. Section 4 focuses on A/B testing in mobile app and how it is different from testing on the web. Section 5 proposes a quasi A/B testing framework to evaluate app release, with results shared from evaluating Voyager [3]. Section 6 concludes with future work.

2. LITERATURE REVIEW

In this section we review the theoretical foundations and applications of A/B testing and quasi A/B testing as well as existing tools and products for mobile testing.

The theory of controlled experiment dates back to Sir Ronald A. Fisher's experiments at the Rothamsted Agricultural Experimental Station in England in the 1920s [6]. Since then, many textbooks and papers from different fields have provided theoretical foundations [7,8] for running controlled experiments, including the most widely adopted Rubin Causal Model [22,23]. While the theory may be straightforward, the deployment and mining of experiments in practice and at scale can be complex and challenging [15]. In particular, several past KDD papers have discussed at length the experimentation systems used at Microsoft Bing, Google, Facebook and LinkedIn [9,10,11,12], including best practices and pitfalls [13,14].

On the other hand, there are many situations where running a controlled experiment is infeasible and one has to study causal impact based on observed data. [21] discusses the assumptions of applying the Rubin Causal Model to real evaluations. [24, 25] show that the Ignorable Treatment Assignment Assumption is associated with the assumptions of OLS regression. Many models have been developed to address the challenges of observational studies, such as propensity score models [21], Heckman's sample selection model [19] and Doubly Robust Estimation [18]. As far as we know, quasi-experiment techniques have been more widely utilized in econometrics and clinical studies [20,26], and there are few studies of natural experiment in the Internet world. A recent work in this area [17] described a framework for estimating causal effect through identifying mediators and its application to advertising.

While we focus on in-house A/B testing solutions for mobile app development, there are companies specializing in mobile A/B testing, such as Apptimize [27], Optimizely [28] and Mixpanel [29]. These companies provide third party platforms and tools that help mobile developers test in mobile. Other than these commercial products, there seems to be very little literature focusing on mobile A/B testing that we are able to find.

3. MOBILE APP RELEASE PROCESS AND ITS IMPLICATIONS

In most online web development, agile software development process is highly promoted. Being able to iterate fast and improve continuously is crucial for both risk minimization and feature development. Most importantly, it is key to ensure the best user experience because any unexpected issues can be fixed or mitigated as soon as possible. Such fast iterations are feasible for

an online website because all changes are controlled on the server side. When a user visits a site, the server pushes the data to the browser to render. New feature releases can happen constantly and continuously without interruption on the end users. In an A/B test, whether the user sees A or B is fully managed by the server and is entirely independent of the end users behavior. Take Linkedin.com as an example. Whether to show a red or yellow button, whether to show a newly revamped homepage or not - these are all changes that can happen instantaneously after server side deployment.

However, feature release process and experimentation in a mobile app are quite different. Instead of the app developers having full control over the deployment and release cycle, the app release process involves all three parties, the app owner (in our case, LinkedIn), the app store (e.g. Google Play or Apple App Store), and the end users. After all the code is ready, the app owner needs to submit a build to the app store for review. Assuming the build passes review (which in the case of iOS takes about a week), releasing it to everyone does not mean that everyone who visits the app will have the new version. Getting the new version is a software upgrade, same as updates we get on our desktops where we can delay or ignore while continuing to use the old version. Some end users take weeks to adopt.

It is worth pointing out that Google and Apple have very different review policies. While it takes about a week to go through Apple's review process, Google Play store's review is almost instantaneous [1]. In addition, Google allows staged roll out of a new version [2], while Apple app store only supports full roll out. Such differences have important implication on app development and evaluation, as we will cover in Sections 4 and 5.

For most web facing companies who truly believe in agile development, it is painful to have to wait for 2+ weeks for a change to take effect, especially when our end users are dealing with a buggy app. What's worse, depending on how fast users adopt new versions, it may take even longer to reach more than 90% users. We have more details on the adoption process in Section 5.1. Moreover, even if new app versions can be built and released every other week, it is annoying from the users' perspective to have to constantly update the app.

This is not to say that we cannot A/B test on mobile. As a matter of fact, there are by and large three kinds of experiments that we can conduct on mobile app.

1. **Server-side changes**: There are many features on mobile that are backend driven. Relevance features are good examples in this category. Experiments on such changes are conducted the same way as experiments on the web [9,10]. Because the code changes happen on the server side only, it only takes server side deployment for the changes to take effect, which can happen independent of the app release.
2. **Client-side changes**: There are many features that need to be controlled from the app itself, including, for example, the look and feel of a button and the number of feed items to fetch from the backend at a time. Any code changes on these features need to be coupled with an app release, and hence are subject to the potential delay. Because of this, there are quite some differences when conducting A/B tests on the app itself. We have devoted Section 4 to go into depth on this topic.
3. **Big changes that cannot be A/B tested**: There are cases where many changes have to be bundled together and it is impossible to put them all behind an A/B test due to infrastructure changes and limitations. For example, LinkedIn recently rewrote the entire flagship app (Project Voyager [3]) and we were not able to experiment the new app as a whole. In the web world, if we cannot split the traffic to two different versions of the site, we are out of luck - once we flip the switch to the new site, we lose the ability to measure the performance of the old site side-by-side. The only comparison that is possible between the new & old sites is a before & after comparison, which is known to suffer from confounding effects such as seasonality. However, for mobile app releases, because not all users adopt the new version at the same time, there is a period of time where we have both versions of the app serving real users. This gives us an opportunity to conduct a quasi-experiment. Details are discussed in Section 5.

4. MOBILE A/B TESTING

As we have mentioned in Section 3, whenever possible, features on mobile apps are tested through controlled experiments. Since server-side changes are usually experimented the same way on app and web, and we have already shared in details how web A/B tests are conducted at LinkedIn in KDD'15 [12], we focus our discussion in this section on the kind of experiments that are more specific to mobile - experiments on the client-side features. Recall that these are features that are controlled from the app, and any code changes on these features require a new app release.

The architecture for a web-based A/B testing system usually has the three essential components, also corresponding to the three steps in an A/B testing process: (1) design (2) deployment (3) offline analysis [9,12]. The same three components are needed for a mobile A/B test. Therefore, we follow a similar structure for our discussion in this section. We focus on highlighting the key differences between app and web testing, while leaving the similarities for readers to follow up in other papers [10, 11].

4.1 Design

Experimental design is arguably the most important step in the testing workflow to get good and meaningful results. As Sir R. A. Fisher put it [13] "To consult the statistician after an experiment is finished is often merely to ask him to conduct a post mortem examination. He can perhaps say what the experiment died of." To be more concrete, design is usually the phase when we determine what we want to experiment on, the goal, the targeted population, the traffic split among variants, and the duration of the experiment.

As described in Section 3, because of the constraint with the mobile app release process (new code cannot be shipped to end users easily), A/B testing on any client-side changes needs to be "planned ahead". In other words, all experiments, including all the variants for each of these experiments, need to be coded and shipped with the current app build. Any new variants, including bug fixes on any existing variant, have to wait for the next app release. This has the following two implications: (1) A/B testing has become more extensive. Whenever possible, new features are always rolled out under A/B tests. One essential function of A/B tests is risk minimization. It allows us to roll out a new feature to a small, randomized user group to evaluate. More importantly, in a mobile world, if a feature doesn't perform as we intended, we can instantly revert back to an equivalent of the older app version by shutting down the traffic to the test, without having to go through a client release cycle (which can take weeks for slowly adopting users). The benefit of preventing our end users from being stuck with a faulty app for weeks really helps promote a

truly "test everything" culture among our developers. (2) Parameterization is used extensively to allow flexibility in creating new variants without a client release. This is because even though new code cannot be pushed to the client easily, new configurations can be passed in payload, which effectively creates a new variant as long as the client understands how to parse the configurations. For instance, we want to experiment on the number of feed items to fetch from the backend at a time. We can put our best guesses in the code and experiment only with what we have planned in place, or we can parameterize the number and have the freedom to experiment on any numbers after the release.

Real time targeting is more prevalent in mobile experiments, as the three most widely used targeting attributes (platform, operating systems and app version) are only available at runtime. Many features are unique to a particular platform and OS. Moreover, the same feature usually performs very differently between tablet and phone, and between iPhone and Android. Therefore, when designing a mobile experiment, we usually need to consider targeting a specific platform and OS combination. In addition, many experiments only exist in certain app versions. Obviously, newly added experiments do not apply to older app versions. At the same time, some experiments may only exist in older versions. It is important to target the correct version when rolling out an experiment.

4.2 Deployment

After design is complete, deploying an experiment in web setting usually just involves two components, the application layer that implements the alternative variant behavior according to experiment assignment, and the service layer that is a distributed cache and experiment definition provider [12]. In the mobile setting, the application layer is logically broken into two parts:

1. Server side: It communicates with the service layer, passes experiment assignment information to the mobile client, and logs the experiment event.
2. Mobile client side: It periodically fetches the experiment assignment information from the server and executes it accordingly.

As we mentioned earlier, the experiment implementation on the mobile client side needs to be shipped with the new app version. Once that is done, we can activate the experiment to enable a small percentage of traffic on treatment. Similar to web applications, the activation information propagates from the service layer to the application server every 5 minutes. However, this does not mean the experiment is fully deployed. As a matter of fact, the deployment is usually delayed on the mobile client by at least one user session. This is because new assignment information is usually only fetched at the beginning of each session. In addition, these new assignment usually does not take effect till the next session. This is because we do not want to change a user's experience after they have started the session already. For heavy users who visit multiple sessions a day, such delay is small, but for light users visiting once a week, the experiment does not actually take place till a week later. Such delay and how it depends on user's visitation frequency imply that not only does the signal appear to be weaker at the beginning of the experiment, it is also more biased towards reactions from heavy users.

Moreover, to reduce the number of communications with the server, assignment information is fetched for all active experiments at once and then cached on the mobile client. This is the case regardless of whether an experiment is actually triggered or not. For example, there is an experiment that only affects users who visit the "Me" tab. However, the assignment for that experiment will still be fetched even if a user doesn't visit "Me" during that session. Because experiment information is tracked on the server side, such "over fetching" means we are marking more users as affected by the experiment than there actually are, effectively diluting the signal when we want to evaluate the "real" impact. One potential fix to this problem is to only track experiment event after an experiment assignment is actually used, not when it is fetched.

4.3 Offline Analysis

By and large, the analysis of mobile app A/B tests is done similarly as web experiments. As mentioned in Section 4.1, experiments can be ramped differently for different platforms and operating systems. Even experiments that are ramped uniformly across the population, the performance in these segments can be drastically different. In many experiments we have conducted, it is not unusual to see a positive lift on one platform and a negative impact on the other. It is therefore important to always drill down to examine the per-segment impact in addition to the overall site-wide impact.

Besides the dilution and the delay effect discussed in Section 4.2, one thing to watch out for is the potential interactions between different user interfaces. Many of our members access LinkedIn through desktop, mobile app and mobile web. While it is unusual to drive traffic between desktop and mobile in a session or a day (we don't expect to change user's mobile usage drastically in a short period of time), it is, however, common to shift traffic between mobile app and mobile web. For example, many mobile users visit LinkedIn via clicking on their emails. The email link can either take the user directly to the app (assuming he or she has the app installed), or to the mobile web. When analyzing an experiment, it is important to know whether an experiment may cause or suffer from such an interaction, in which case, we cannot evaluate app performance in isolation, but need to look at user behavior holistically on both mobile app and mobile web. In addition, because user experience on the app tends to be better, directing traffic from the app to the web tends to bring down the total engagement, which may be a confounding effect not intended by the experiment itself.

5. MOBILE QUASI A/B TESTING

We have so far highlighted the special challenges when conducting randomized experiments on mobile app compared with web testing. Most of the differences result from the fact that mobile app release cycle involves not just the app owner, but also app stores and end users. On the other hand, the exact same differences also create an opportunity to evaluate changes that we are not able to A/B test on. These tend to be big changes that involve fundamental infrastructure improvements and holistic product redesign. One such example is LinkedIn's recent release of the new flagship app (Project Voyager [3]).

To be more concrete, the opportunity to evaluate such big releases exists because of the following: First, users take time to adopt the new app. The adoption period creates a natural experiment. Second, Google supports staged roll out of an app. Even though neither enables a fully randomized experiment, we can leverage both to measure the performance of a new app version through a quasi-experimental study. As we will lay out in this section, such quasi-experiment, coupled with thorough validation, is able to estimate the impact from a new app release with little bias, as demonstrated by results from our recent Voyager launch.

This section is organized as follows. We start in Section 5.1 with a deep dive into how mobile app adoption works, who the adopters are and how the adopter group evolves over time. This is crucial in helping us come up with quasi-experimental techniques to adjust for adoption bias. Section 5.2 reviews the basic quasi-experiment methods in the context of evaluating app release. This is not meant to be a comprehensive literature review, but to establish the notation and building blocks for the next section. In the final Section 5.3, we go into depth discussing and comparing the techniques we use to evaluate the Voyager release. Separate methods are needed for iOS and Android, as the Android release was a 20% staged roll out during the evaluation period.

5.1 Mobile App Adoption

Every new app version takes a while to propagate through the user population, as it requires upgrade from the end users. Both Android and iOS give users a lot of flexibility in managing how they want to upgrade. By and large, there are three options: (1) automatically update (2) automatically update only if there is Wi-Fi access (3) manually update. Obviously the choice affects the probability to adopt. The default setting of both operating systems is option (2), with the difference that Android notifies users after the update completes, but iOS updates quietly in background.

The choice of the update options and the ease of access to Wi-Fi are the two most important variables that impact the likelihood of adoption over time. To add to the complexity, we have no visibility into whether a user has updated to the new version until he or she visits the app. One can think of modeling such mechanisms and relationships with a Latent Factor Model (with missing data) to reveal and understand these underlying variables [30]. Since our focus is to evaluate the app performance by adjusting for the intrinsic adoption bias, we instead look to understand who these adopters are and what the key differences are between the adopters and non-adopters.

5.1.1 Adoption Over Time

Because our member populations are intrinsically different across OS, we observe very different adoption curves between Android and iPhone (Figure 1, Left). Adoption is much faster on iPhone, especially during the first couple of days. As we can see in Figure 1, the adoption rate that takes Android a week to reach takes only about three days on iPhone. In general, adoption tends to slow down after a week on both OS. There are some people who stays on the older version for weeks or months before they adopt, and they tend to be the same group of users for each release, likely those who select manual update option on their device. Note that even though the plot only uses data from the recent major release with significant PR pushes, almost exact same plots are observed from other minor historical releases, indicating little social effects on adoption of newer app versions.

From the adoption mechanism described above, it is not surprising to see that adopters tend to consistently adopt through multiple historical app releases. As shown in Figure 1 (Right), a previous adopter is in general 2 – 3 times more likely to adopt again than a previous non-adopter. Note that the risk ratio tends to be smaller in the earlier days. This indicates that historical adoption pattern is less predictive of future adoption shortly after launch. This can be explained by the fact that whether a user adopts or not is largely driven by his or her choice of auto-update options and access to Wi-Fi. Conditional on the same user, we can treat the Wi-Fi availability as a random process independent of users' visit to LinkedIn. Therefore, whether a user adopts *shortly after release* has a large random component and is hard to predict based on historical adoptions.

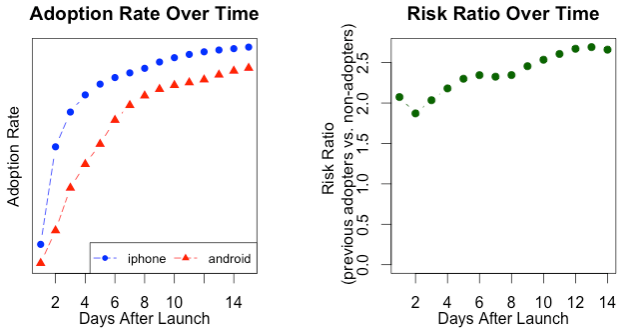

Figure 1. Left: adoption rate over time. Right: the probability of adopting in current release is higher for previous adopters than for previous non-adopters.

5.1.2 Adopters vs. Non-adopters

Among the adopters, there is a big difference between those who adopt on day one and those who adopt on day 14. We group the adopters into cohort 1 through 14 depending on the day they adopt. As show in the heatmap (Figure 2), it is clear that users who adopt earlier are more engaged than the users who adopt later. Also note that even though the diagonal has higher values, it is not an indication that users are more engaged on adoption day. It is an artifact that we don't know users' adoption status until they visit the app, and hence by definition users are active on their adoption day.

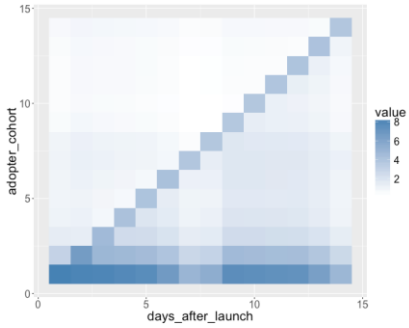

Figure 2. Engagement by adopter cohort. Earlier adopters are more engaged.

Now let's turn to adopter and non-adopter differences. Note that both adopters and non-adopters are active during the analysis period, and their adoption labels can change over time as more people adopt. Figure 3 (Left) plots the difference as percentage delta of a key engagement metric between adopters and non-adopters. The data used is from a minor historical release that is not supposed to create any actual impact on metrics. Interestingly enough, when the analysis period is a single day, the difference starts at over 50%, then drastically decreases to below 10% after a week. However, if we look at cumulative differences cross day, the percentage delta stays at above 30% throughout the entire week! Similar trend is observed for almost all metrics we looked at. It turns out adopters tend to be active on more days. So if we accumulate their activities over time, the adoption bias is bigger. This can be explained by the simple equation below, where the cumulative metric can be decomposed as users' engagement on an active day with a multiplier of total active days. Indeed, by day seven, an average adopter is active on 62% more days, explaining the majority of the difference in the cumulative engagement.

$$\frac{Engagement}{UU} = \frac{Engagement}{ActiveDays} * \frac{ActiveDays}{UU}$$

There are two implications. First, without modeling for adoption bias, we can effectively mitigate it by comparing adopters and non-adopters in a smaller time window. However it assumes the number of visit days is not impacted by the treatment, which should be tested separately. Second, as shown in Figure 3 (Right), if we control for the days users are active, the difference between adopters and non-adopters become much smaller (from over 50% to less than 20% for day one), indicating historical active days is a strong covariate for adjusting adoption bias.

It is worth noting that some users take more than two weeks to adopt, but they are as active as the adopters. These are likely users with auto-update off. It seems that users' choice of auto-update option is fairly independent of their engagement with LinkedIn.

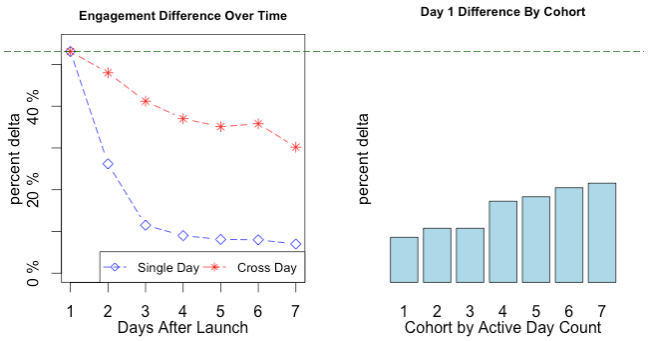

Figure 3. Left: Engagement differences between adopters and non-adopters (A/A). Single-day difference decreases drastically over time. Right: The within cohort differences between adopters and non-adopters are much smaller.

5.2 Common Quasi-Experiment Techniques

As we have already seen in Section 5.1, there are big differences between adopters and non-adopters. Simply comparing those who are on the new app with those on the old app will give a misleading estimate of treatment effect. In this section, we evaluate common statistical techniques to reduce adoption bias; in particular, methods based on OLS regression and propensity score models.

5.2.1 OLS Based Methods

Assume our metric of interest follows a linear model:

$$\boldsymbol{Y} = \beta_0 + \boldsymbol{Z}\beta_1 + \boldsymbol{X}\boldsymbol{\beta_2} + \boldsymbol{\epsilon} \quad (1)$$

where $\boldsymbol{Y} = (Y_1, \dots Y_n)^T$ is the outcome vector for users $i = 1, \dots, n$, $\boldsymbol{Z} = (Z_1, \dots Z_n)^T$ is the dichotomous variable of treatment, and $\boldsymbol{X} = (\boldsymbol{X_1}, \dots \boldsymbol{X_n})^T$ is a matrix representing all the covariates that correlates with both $\boldsymbol{Z}$ and $\boldsymbol{Y}$ (e.g. historical engagement metrics). Under this model, the coefficient β_1 is the ATE under the Rubin Causal framework. If the covariates $\boldsymbol{X}$ are omitted, the OLS estimator of β_1 can be biased, and

$$bias = (\boldsymbol{D}^T\boldsymbol{D})^{-1}\boldsymbol{D}^T\boldsymbol{X}\boldsymbol{\beta_2} \quad (2)$$

where $\boldsymbol{D} = (\boldsymbol{1}, \boldsymbol{Z})$. Note that the bias is zero only if $\mathbf{X}$ is independent of $\mathbf{D}$ (so $\boldsymbol{D}^T\boldsymbol{X} = \boldsymbol{0}$, which is true for randomized experiments) or that $\mathbf{X}$ does not predict $\mathbf{Y}$ (so $\boldsymbol{\beta_2} = \boldsymbol{0}$).

Practically such bias gets smaller if major covariates that confound the treatment assignment ($\boldsymbol{Z}$) and outcome variable ($\boldsymbol{Y}$) are included in the regression model. However, it is not possible to prove that all non-ignorable covariates are included. In our applications, we are able to evaluate the models based on a validation framework to be discussed in Section 5.3.

There are some potential improvements on the OLS model:

1. It is unlikely that the relationship between the response variable and the covariates is linear. We can consider applying Box-Cox transformation [31] before fitting the model.
2. We can generalize the treatment model (1) to an endogenous switching model by allowing for interactions between the treatment variable and the covariates. The model can then be rewritten as

$$\boldsymbol{Y_1} = \boldsymbol{X_1}\boldsymbol{\beta_1} + \boldsymbol{\epsilon_1} \ if \ Z = 1 \quad (3)$$

$$\boldsymbol{Y_0} = \boldsymbol{X_0}\boldsymbol{\beta_0} + \boldsymbol{\epsilon_0} \ if \ Z = 0 \quad (4)$$

where two equations are fitted separately using data from adopters and non-adopters respectively.

Interestingly, looking at Equation (3) alone, the problem becomes a classical missing value problem where non-adopters are clearly missing at "non-random". Therefore, we can also apply many techniques that address missing value problems to estimate Equation (3) and (4) separately. The method we considered in Section 5.3 is the one proposed by Heckman [19], which models the selection process based on probit regression and assumes that the error term in the probit model and the linear regression are bivariate normally distributed.

5.2.2 Propensity Score Based Methods

Another family of approaches that have been developed to correct for selection bias is through the propensity score, i.e. the probability of receiving treatment. The propensity score is usually estimated through a logistic regression, defined as

$$P(Z = 1|X = x) = \frac{e^{\boldsymbol{x\beta}}}{1 + e^{\boldsymbol{x\beta}}}$$

The score can be used for

1. **Matching or subclassification.** The intuition is that samples with similar propensity scores can be considered as counterfactuals. By reducing to a one-dimensional score, one can incorporate potentially many covariates that have predictive power on treatment selection and still obtain reasonable matched sample sizes.
2. **Weighting.** With the Rubin-causal counterfactual framework, one can show that given the true probability of being treated we have $E(\frac{Y_i}{P(Z_i=1)}) = E(Y_{i1})$ [16]. Hence we can use $\frac{1}{P(Z_i = 1|X)}$ as weight for adopters (and $\frac{1}{1-P(Z_i = 1|X)}$ as weight for non-adopters). However, such inverse weighing mechanism assumes that the propensity score is the true probability of being treated. A more recent class of estimators combines the OLS method and propensity score inverse weighing models. Such Doubly Robust Estimators are consistent as long as at least one of the linear model or the propensity score model is correctly specified [18].

5.3 Evaluating Voyager Release

As we have mentioned earlier, LinkedIn has recently redesigned and rebuilt our flagship app from scratch [3]. The new app, code name "Voyager", should be considerably more intuitive and useful. One key challenge we faced was how to evaluate whether users indeed are benefiting from using the new app. If we were able to A/B test the app, it would have been an easy question to answer. However, with the drastic changes on both backend and frontend infrastructure, it is impossible to test all the changes in a randomized, controlled setting. The closest to A/B test we can get

is to leverage Google Play store's staged roll out functionality. But even with the staged roll out, we still have to face adoption bias, and therefore cannot analyze the results as a simple A/B comparison.

As we mentioned earlier, the biggest challenge of any quasi-experimental method is how to validate that the causal relationship is indeed true and not due to some omitted variables in the model. Fortunately, we are able to have a validation framework based on historical data. Historically, we release new app versions to app stores periodically. These regular releases usually ship no more than minor bug fixes, which create a unique opportunity to (1) understand the adoption behavior (section 5.1) and consequently utilize it to build better quasi experiment models; (2) evaluate the validity of a model. These historical releases essentially serve as A/A tests, as it is reasonable to assume that such incremental releases only cause minor impact and the true average treatment effect (ATE) is close to 0. In addition, we have also observed that those validation results hold consistently across multiple historical versions.

Unless specified, all results discussed from here on are cumulative up till a specific day. Even though single-day results tend to suffer less from adoption bias (as seen in Section 5.1), it makes the assumption that the new app does not make users visit more (or less) days. With a release as big as Voyager, it is not a plausible assumption.

Because of the differences in how we roll out the app in iOS and in Android, we have to use different quasi A/B methodologies to evaluate iOS and Android separately, which we share in sections below.

5.3.1 iOS

Unlike in Android, there is no opportunity to release an app version to a randomized percent of users in iOS. This makes the adoption process fully observational. Evaluating Voyager in iOS becomes a "classical" quasi-experiment problem and existing techniques discussed in Section 5.2 should apply directly with only minor improvements. In this section, we compute the bias of each method during the week after release (in terms of percentage delta) for a key engagement metric. To make the comparison fair, the same set of covariates are used across methods. All results are based on a minor historical iOS release (an A/A release).

Before we get to the comparison results, there are a couple of key learning from building the quasi-experiment models.

First of all, many non-adopters become adopters after a couple of days. As we have observed in Section 5.1, whether a user adopts *shortly after release* is somewhat random due to Wi-Fi availability. Such random perturbation creates samples that are very similar in terms of covariates, but with opposite adoption status. These samples confuse the propensity model and make it harder to predict, as we can see in Figure 4 (Left). The baseline AUC is only at 0.65 in the first day compared with 0.82 on day seven. This observation leads us to remove from the non-adopter group those users who in fact are very likely to adopt based on historical adoptions. By removing these "random" non-adopters in the early days, we are able to get an AUC score of close to 0.8 even on day one (green curve). Any improvement for the early days is substantially more valuable in the production environment, as it really helps identify problems early and iterate fast.

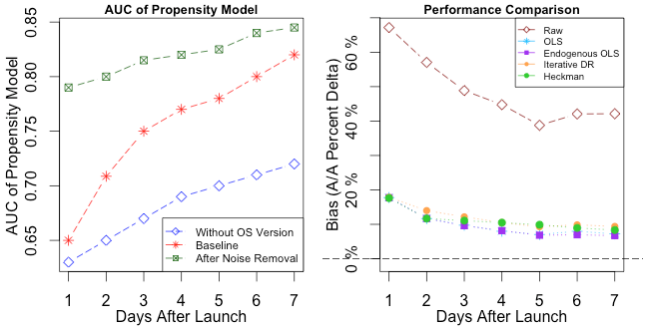

Figure 4. Left: AUC of the propensity model; Right: Bias of various quasi methods evaluated for the iOS release.

Second, it is crucial to include the right features. We have explained the rationale on the choice of some features based on users' adoption behavior in Section 5.1. As directly illustrated in Figure 4 (left), removing one covariate (e.g. OS Version) can decrease the AUC substantially.

As shown in Figure 4 (Right), the raw comparisons are extremely biased, especially in the first days after launch. The amount of adoption bias reduces from about 65% to 40% after a week. The various methods have similar performance over time, with the endogenous OLS model having a slightly smaller bias. Overall, these methods are able to largely reduce adoption bias to about 18% on day one to about 7% on day seven.

5.3.2 Android

Play store makes it easy for app developers to do a staged roll out [2]. We can choose to make the new version available to only a percentage of users. This applies to both existing app users and new users downloading from the Play store. It is a great way to minimize risk and collect feedback from real users. It may not be intuitive at first, but even though these users are randomly selected, we are not able to measure the staged roll out as a regular A/B test. The reason is because from our own tracking, we can only tell whether a user visits with a new app or an old app, but not their actual randomized experiment assignment. As demonstrated in Figure 5, among those visiting with an old app, we cannot differentiate those who are in the control bucket (entire B group) from those who are non-adopters in the "eligible" bucket (A_2 group). If we naively compare members on the new app with members on the old app, we would be comparing A_1 with the rest, and hence suffer from the same adoption bias (or self-selection bias) as a full roll out.

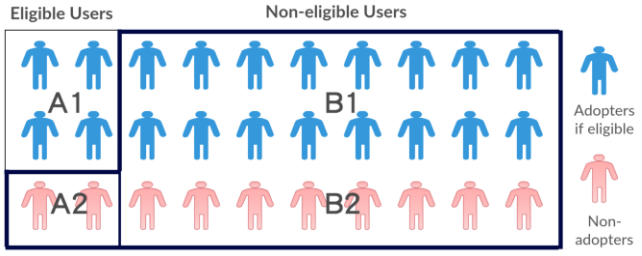

Figure 5. Illustrate how adoption bias exists even in Google staged roll out.

The problem, however, is a little different. In a full roll out, other than the minor random perturbation to the label during the first couple of days (due to Wi-Fi access), the adopters are intrinsically different from non-adopters. Most quasi-experiment techniques described in Section 5.2 try to remove such intrinsic difference by modeling directly on adoption status. For example, propensity

score models build a logistic regression using adoption status as response. When the actual adoption status is determined by a combination of randomization and user intrinsic behavior, such models work poorly. This is because the signal that comes with the status is weak. The B_1 users who were non-adopters can become adopters with simply a reshuffle, which obviously cannot be modeled by any user variables.

At the same time, there is a bright side to the unique situation. In a 20% roll out, for every adopter we expect there to be four non-adopters who are similar to him. If we have a methodology that can identify those "would-be" adopters, we are all set. To successfully reduce adoption bias, such method needs to have an extremely low false positive rate, even if that leads to more false negatives. We illustrate the idea in the following.

Assume we have a selection criteria S to pick out potential adopters. Apply such criteria to the two groups of users observed (A_1 vs. $A_2 + B_1 + B_2$). With high false positives and zero false negatives, we have $S(A_1) = A_1$, and

$$S(A_2 + B_1 + B_2) = S(A_2) + B_1 + S(B_2) > B_1$$

Since A_1 and B_1 are the two comparable user groups, such selection criteria create bias. The higher the false positive rate is, the worse the bias gets. On the other hand, if we have only false negatives and no false positives, we get $S(A_1) \subset A_1$, and

$$S(A_2 + B_1 + B_2) = S(B_1) \subset B_1$$

Since A_1 and B_1 are comparable, $S(A_1)$ and $S(B_1)$ are comparable as well. As long as the false negative rate is not too high and $S(A_1)$ is still a representative subset of A_1 (i.e. out-of-sample bias is small), the comparison between $S(A_1)$ and $S(B_1)$ is meaningful. In Section 5.3.2.1 and 5.3.2.2 below, we propose two methods that are guided by such intuition.

5.3.2.1 Geometric Distribution Model

The idea is to separate the "would-be" adopters (B_1) from the rest of the non-adopters by directly modeling their adoption probabilities. By selecting only users with high propensity to adopt in both adopter and non-adopter groups, we hope to achieve high precision in identifying comparable users. Note that different from the usual propensity score modeling, we only use historical data to model the adoption probabilities. The actual adoption status itself is only used in the final step as treatment assignment label.

As we discussed in Section 5.1, the adoption probability increases as time goes by. We therefore need to model p_{it}, the probability for member i to adopt on day t. As described in section 5.1, we do not know whether a member has adopted the new app version until he or she visits our app. To simplify the problem, we assume that every member i has a chance of p_i to show up as an adopter each day they visit, and that the probability of adopting on day t follows a geometric distribution

$$p_{it} = (1 - p_i)^{a_{it}-1} p_i$$

where a_{it} is the number of active days a user has had by day t.

Based on data from historical adoptions, we can compute the maximum likelihood estimator (MLE) for p_i to be

$$\hat{p}_i = \underset{p_i}{\operatorname{argmax}} \prod_{j=1}^{s} (1 - p_i)^{k_{ij}-1} p_i^{I_{ij}} = \frac{\sum_j I_{ij}}{\sum_j I_{ij} + \sum_j k_{ij}}$$

where k_{ij} is the number of active days user i had before adopting historical app version j, with $j = 1, 2 \ldots s$, and I_{ij} be the indicator function of whether user i adopted version j. Note that in our application, we capped k_{ij} at 14 days, so if user i did not adopt by day 14 for version j, we have $k_{ij} = 14$ and $I_{ij} = 0$.

Finally, on each day t after Voyager release, we compute the probability that user i adopts *by* day t to be $1 - (1 - \hat{p}_i)^{a_{it}}$, which is the cumulative probability based on the actual number of active days we observe. To get the treatment effect, we only select users with $1 - (1 - \hat{p}_i)^{a_{it}} >$ threshold. The threshold is chosen based on A/A cross-validation results.

The results (Figure 6, Left) show that such method is able to correct for almost all the selection bias towards the end of the adoption week. However, regardless of how high a threshold we pick, the precision in the first few days is poor. This echoes our analysis in Section 5.1 where we observed that adoption in the first days is hard to predict.

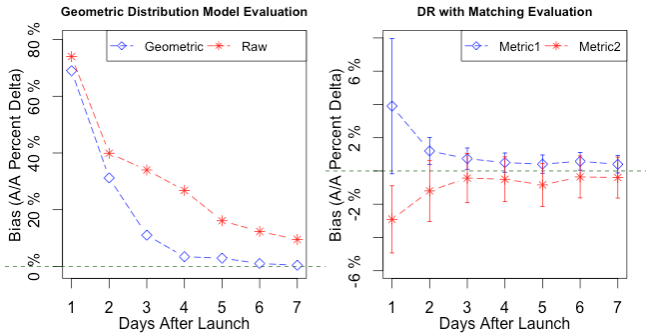

Figure 6. Left: Bias of the geometric distribution model. Right: Bias of the doubly robust with matching model. Both are for Android 20% roll out.

5.3.2.2 Doubly Robust with Matching

As we mentioned early, propensity score models do not work well in the randomized setting. However, because there is supposed to be more users in the non-adopter group who are similar to the adopters, matching directly on covariates themselves is supposed to be an easier task than it is in the full roll out case (as in iOS). There are various methods that perform matching based on covariates, and they all trace back to two that are the most fundamental:

1. **Exact matching**: This is the technique that matches each adopted user to all possible non-adopters with exactly the same values on all the covariates.
2. **Nearest neighbor matching**: This method selects the non-adopters who are closest to each individual in the adopter group in terms of a distance measure specified. The selection process can be done in a couple of ways: (1) a "local" greedy approach that chooses the closest match one at a time, in which case it matters which adopter gets processed first; (2) a "global" optimal approach that finds the matched users with the smallest average absolute distance across all the matched pairs [2].

Neither the local nor the global nearest neighbor approaches can be easily computed in a parallelizable fashion, which makes scaling it to millions of users a hard problem of its own. In addition, on a smaller test data set we tried, the nearest neighbor technique tends to under perform, and highly affected by the distance measure chosen, particularly because we have many categorical covariates. Exact matching, on the other hand, does not work well when there are many covariates (or when some

covariates can take many values, such as continuous variables), as it becomes impossible to find sufficient exact matches. Continuous or ordinal variables have an additional challenge in exact matching because of the lack of distance measures. A difference of 1 pageview is treated the same as a difference of 1000 pageviews, which is clearly suboptimal.

Because of these challenges, we have decided to go with a "Doubly Robust" approach [18], where we fit an exact matching model first and then fits a linear regression model on the matched user sets. The exact matching takes in only about 10 important covariates with the continuous variables carefully bucketized to ensure sufficient matched samples. The regression model has a lot more covariates, including several continuous variables, which can compensate for the somewhat coarse matching and offers more granularity into the covariates. Because of its good performance in validation from first day after release, this is the approach we used in production.

Here are the steps we took for the Doubly Robust estimation:

1. Only variables that cannot be impacted by the treatment itself are included as covariates, including variables collected before the new app version release or stable member attributes such as country or language.
2. For all the variables, ensure common support in both adopter and non-adopter groups by pruning the observations where the empirical densities do not overlap. This is to avoid extrapolation.
3. Select a small set of variables (about 10) to be used for exact matching. We first select representative variables to cover a wide range of member attributes and engagement characteristics. Further reduction is then done by dropping one variable at a time and is cross-validated based on A/A test results (described earlier). The goal is to produce as many matched samples as possible without increasing bias.
4. Covariates used in exact matching are bucketized to reduce their cardinalities. We have noticed that using quantiles to bucketize does not work well for a lot of covariates because of their skewed distributions. For example, the difference between the 10th and the 20th percentiles of several variables has no practical significance. Artificially treating them as two entirely different buckets creates inefficiency when matching the samples.
5. Feed the matched samples into a weighted linear regression model, using the weights produced from the exact matching. We use the endogenous switching model described in Equation (3) and (4), building one model for adopters and one for non-adopters. The models are then used to predict on the entire matched user samples. For example, the model trained on adopters is used to produce $\hat{y}_i^{(1)}$, the estimated response under the new version (treatment), for both adopters and non-adopters. The final doubly robust estimator of the treatment effect is a weighted average difference between a world where everyone has adopted and a world where no one has adopted:

$$DRE = \frac{1}{\left(\sum_{i=1}^{n} w_i\right)}\left(\sum_{i=1}^{m} w_i\left(y_i - \hat{y}_i^{(0)}\right) + \sum_{i=m+1}^{n} w_i\left(\hat{y}_i^{(1)} - y_i\right)\right)$$

where w_i is the weight from matching and y_i is the observed value. WLOG we assume users $i = 1, \ldots, m$ are adopters.

We have noticed that it is important to have sufficient number of samples that are matched. During the first day post launch, as the number of adopters is still relatively small (especially in 20% roll out), the set of variables we include in the matching step is much smaller. As a result, we matched about 60% of adopters on day one, while that number reduces to 20% after day one. On the other hand, the matched non-adopters went from about 5% to 10%. The median number of non-adopters matched per adopter is 3.

The validation results look very positive. As shown in Figure 6 (Right), for the two key metrics we tracked, we had only about 3 – 4 % bias on the first day, compared with an over 70% bias in the raw comparison shown in the left plot. The bias reduces to less than 1% after day three.

5.3.3 Novelty Effect

For big releases that involve drastically different user experience, we usually expect strong novelty effect, as users tend to explore the new experience more at the beginning. One obvious question when it comes to evaluating the performance of a new app version is: is there novelty effect and if so, how long does it take for it to go away? In a randomized A/B test, a simple and practical approach is to check whether the daily treatment effect, measured as the percentage delta between treatment and control, dies down as days go by. With adoption bias, novelty effect is confounded by the fact that people who adopt earlier are simply different from people who adopt later. The quasi-A/B framework in Section 5.3.2.2 provides us with a way to separate out the inconsistent adoption bias and the novelty effect.

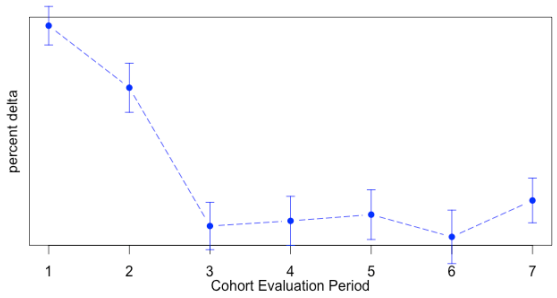

Figure 7. Novelty effect goes away after the first couple of days.

We start with identifying two cohorts of users: the adopter cohort and the non-adopter cohort. Obviously, if we construct the cohorts on day one, we will have some of the non-adopters converted to be adopters during the period of evaluation. Because this is a retrospective study, we can overcome this problem by removing users who ended up adopting during this period from the non-adopter cohort (we can safely assume there is no conversion the other way around, going from adopter to non-adopter). As a second step, we match the cohorts using the same exact matching method as outlined in Section 5.3.2.2. This gives us two matched cohorts with weights. Finally, we remove further bias by applying a weighted linear regression on the matched cohorts as in the doubly robust approach. It is important to note that even though the matched cohorts are constructed once, the evaluation is done on the same cohorts daily, regardless whether a cohort user is active in that particular day. As shown in Figure 7, the treatment effect (measured as percent delta) is much larger in the first couple of days and then settles down to a smaller, consistent value, providing compelling evidence that the novelty effect lasted for just a couple of days.

6. SUMMARY AND FUTURE WORK

In this paper we discussed many differences when experimenting on mobile compared with on desktop. We also proposed and established a quasi A/B testing framework to evaluate mobile app

release. Many insights around how users adopt a new app version were shared and discussed.

One very interesting problem that we didn't cover is to understand the latent factors underlying users' adoption behavior (Section 5.1), which should help further improve the quasi models. Moreover, notice how the quasi models have a lot less bias for Android (Figure 6) than they have for iOS (Figure 4). It is because Android was a randomized 20% roll out. If Apple App Store also supports a staged roll out feature, it will greatly benefit not only the app developers, but also app users, as they will get apps that are better evaluated and optimized.

7. ACKNOWLEDGEMENT

The authors wish to thank Bryan Ng, Aarthi Jayaram, Yael Garten and Kiran Prasad for insightful discussions, Romer Rosales and Eytan Bakshy for feedback. We would also like to thank all members of the experimentation team, especially Weitao Duan who was involved in the quasi A/B effort. Finally this work wouldn't have been possible without the guidance and support of Igor Perisic.

8. REFERENCES

[1] Sarah Perez [Online] http://techcrunch.com/2015/03/17/app-submissions-on-google-play-now-reviewed-by-staff-will-include-age-based-ratings/

[2] Google Play [Online] https://support.google.com/googleplay/android-developer/answer/6346149?hl=en&ref_topic=3450989

[3] Jordan Novet "LinkedIn shows off Project Voyager, its new flagship mobile app" [Online] http://venturebeat.com/2015/10/14/linkedin-shows-off-project-voyager-its-new-flagship-mobile-app/

[4] Fisher, Ronald A. Presidential Address. *Sankhya: The Indian Journal of Statistics.* 1938, Vol. 4, 1.

[5] The 2015 U.S. Mobile App Report [Online] https://www.comscore.com/Insights/Presentations-and-Whitepapers/2015/The-2015-US-Mobile-App-Report

[6] Yates, Frank, Sir Ronald Fisher and the Design of Experiments. *Biometrics,* 20(2):307–321, 1964.

[7] Box, George EP, J. Stuart Hunter, and William Gordon Hunter. *Statistics for experimenters: design, innovation, and discovery*. Vol. 2. New York: Wiley-Interscience, 2005.

[8] Gerber, A. S., and Green, D. P. Field Experiments: Design, Analysis, and Interpretation. WW Norton, 2012

[9] Tang, Diane, et al. Overlapping Experiment Infrastructure: More, Better, Faster Experimentation. *Proceedings of the 16th Conference on Knowledge Discovery and Data Mining.* ACM, 2010.

[10] Kohavi, Ron, et al. Online Controlled Experiments at Large Scale. KDD 2013: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining. 2013. http://bit.ly/ExPScale.

[11] Bakshy, Eytan, Eckles, Dean and Bernstein, Michael S. Designing and Deploying Online Field Experiments. *Proceedings of the 23rd international conference on World Wide Web,* pages 283-292, ACM, 2014

[12] Xu, Ya, et al. "From Infrastructure to Culture: A/B Testing Challenges in Large Scale Social Networks." *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015.

[13] Kohavi, Ron, et al. Trustworthy online controlled experiments: Five puzzling outcomes explained. *Proceedings of the 18th Conference on Knowledge Discovery and Data Mining.* 2012, www.exp-platform.com/Pages/PuzzingOutcomesExplained.aspx.

[14] Kohavi, Ron, et al. Seven Rules of Thumb for Web Site Experimenters. KDD 2014: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. 2014.

[15] Kohavi, Ron. et al. Controlled experiments on the web: survey and practical guide. *Data Mining and Knowledge Discovery.* February 2009, Vol. 18, 1, pp. 140-181. http://www.exp-platform.com/Pages/hippo_long.aspx.

[16] Imbens, Guido M., and Jeffrey M. Wooldridge. *Recent developments in the econometrics of program evaluation*. No. w14251. National Bureau of Economic Research, 2008.

[17] Hill, Daniel N., et al. "Measuring Causal Impact of Online Actions via Natural Experiments: Application to Display Advertising." *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015.

[18] Funk, Michele Jonsson, et al. "Doubly robust estimation of causal effects."*American journal of epidemiology* 173.7 (2011): 761-767.

[19] Heckman, James J. "Sample selection bias as a specification error (with an application to the estimation of labor supply functions)." (1977).

[20] Heckman, James J., and Richard Robb. "Alternative methods for evaluating the impact of interventions: An overview." *Journal of econometrics* 30.1 (1985): 239-267.

[21] Rosenbaum, Paul R., and Donald B. Rubin. "The central role of the propensity score in observational studies for causal effects." *Biometrika* 70.1 (1983): 41-55.

[22] Holland, Paul W. "Statistics and causal inference." *Journal of the American statistical Association* 81.396 (1986): 945-960.

[23] Rubin, Donald B. "Estimating causal effects of treatments in randomized and nonrandomized studies." *Journal of educational Psychology* 66.5 (1974): 688.

[24] Kennedy, Peter. *A guide to econometrics*. MIT press, 2003.

[25] Greene, William H. *Econometric analysis*. Pearson Education India, 2003.

[26] Harris, Anthony D., et al. "The use and interpretation of quasi-experimental studies in medical informatics." *Journal of the American Medical Informatics Association* 13.1 (2006): 16-23.

[27] Apptimize [Online] http://apptimize.com/

[28] Optimizely [Online] http://optimizely.com/

[29] Mixpanel [Online] https://mixpanel.com/

[30] Agarwal, Deepak, and Bee-Chung Chen. "Regression-based latent factor models." *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 2009

[31] Box, George EP, and David R. Cox. "An analysis of transformations." *Journal of the Royal Statistical Society. Series B (Methodological)* (1964): 211-252