1. **Predict organic activity rate**: Here, we are estimating the user's organic activity per visit. This is done by computing the number of organic stories they have had in the past, normalized by the number of times they have visited the feed. Intuitively, this should be proportional to the number of followed users: the more users one follows, the more organic activity they will have, on average.

2. **Predict visit rate**: This is a simple calculation that averages the number of times a user has visited the AF over a set date range.

With these predicted rates, we can easily estimate 1) the number of times the user will visit the AF in the next day (which is how far out our nightly content stream computations go, keeping in mind that this number will be less than 1 for most users), and 2) the number of organic stories they would see on each visit. From these numbers, we can determine the number of generated stories that must be issued per visit, in order to hit the 10 new story minimum. For each of the generated stories, we can also assign timestamps such that they are evenly issued throughout the relevant time range, beginning from the time of the user's last visit to the AF to the end of the next day (which is as far into the future that the content stream job is responsible for computing). Figure 3 illustrates how, in most cases, issuing generated stories at uniform time intervals intersperses the generated content within the organic stories naturally. As a result, we have populated content streams with various generated stories that are ready to be delivered.

4.3 Storing Content Streams

After generating content streams on Hadoop, we need a way to bring them to the wider web. For this purpose, we use an intermediate datastore. Datastores for content streams have two primary constraints. We are adding an additional step to the feed creation process on the web side, and thus a slow content stream fetch negatively impacts user experience as they would be expected to wait. This gives rise to our first constraint: the datastore must be performant. Content streams contain data that span a wide range of time. We only care about a small portion of this data at any point, which gives rise to our second constraint: the datastore must be able to efficiently filter content streams before passing them to the client.

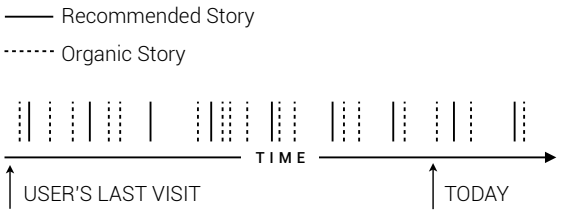

Figure 3: A content stream is computed daily for each user, and is responsible for serving generated stories that are timestamped from the user's last visit to the end of the next day. This figure shows a hypothetical timeline along which a user's stories occur. Note that the generated content (which are timestamped at uniform intervals) is nicely interspersed with their organic content.

Redis is one datastore that meets these constraints. Redis is an in-memory datastore, optimized for speed. It acts much like a key-value store, but boasts support for various data structures. Of particular interest are ZSets, Redis' implementation of a scored, sorted set. Content streams' structure - a timestamp and associated value - lend themselves naturally to ZSets. Redis also allows range queries on ZSets, allowing us to select values by score (which, in our case, is the timestamp for which the story should appear on a user's feed). These queries are performant: 90% of queries return in under 250 microseconds. Compared to the overall AF render, which is around 300 milliseconds, this is quite fast. Thus, Redis provides an efficient go-between for our data generation on the Hadoop side, and our data consumption on the web side.

4.4 Integrating with the Activity Feed

With easy access to our content streams, all that remains is integration with the AF on the web. This step happens when the user visits Etsy, and is responsible for delivering content to the user. Since Etsy's AFis a pull-based architecture, a feed is rendered by first "pulling" (or querying) content from all sources relevant to a particular user. This step is called *aggregation*. Every node in a user's immediate social graph is treated as a content producer. Each node is queried for new content, which is then appended to the user's feed. This is where content streams are integrated.

Content streams are treated like any node in a user's social graph, and queried for new content. Redis's ZSet query support is helpful here: `zrangebyscore` queries are used to find all content with a score greater than the timestamp of last aggregation, but less than our current timestamp. This means that generated content will be delivered only at its prescribed timestamp, and not before. This content is then appended to the feed (sorted by timestamp) like any other content. As noted, this usage of pre-assigned timestamps allows us to naturally intersperse generated stories amongst a user's organic stories, avoiding large bursts of unfamiliar content that might overwhelm the user.

Once a feed has been aggregated, it must be rendered. Prior research has shown that recommendations perform best when presented with context. An example of context may be "based on items you viewed, you may also like this." Here, context is tied to the system that is responsible for generating the targeted content - this may be a recommender system or an advertising engine. The render step consists of linking each piece of targeted content with its source, and passing it off to the feed's standard render process for display on the screen. At this point, our generated content and organic content have been woven together to form a coherent and vibrant feed experience, much like the one shown in Figure 1. AF content can now be consumed on any device, be it through desktop, mobile web, or mobile apps.

5. EXPERIMENTS

Adding content to a feed necessarily affects all other content surrounding it - generated content could overwhelm organic content, making high-value organic content difficult to find. Generated content may also be perceived as low-quality and mis-targeted. Because of the wide variability of feeds in both content sources and quantity of content,

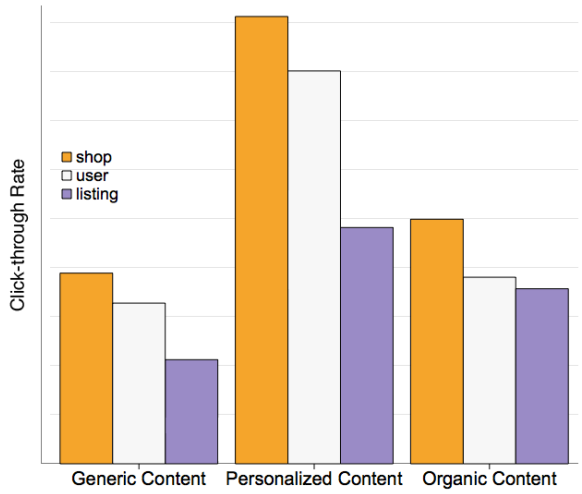

Figure 4: A break-down of CTR for different types of generic, personalized, and organic stories. It is clear that personalized content best engages users.

designing a heuristic to evaluate the quality of all feeds is difficult. Thus, we turn to experimentation to understand the effects content streams have on users.

Etsy's experimental framework is based on *A/B experimentation*, where we select key metrics to monitor as groups of users interact within control or treatment variants. The metrics we chose were: listing view rate, listing favorite rate, and shop favorite rate, all of which are measured on a per-visit basis. These are all core actions that not only lead to purchases, but also allow improvement in the generation of our targeted content. We also monitored the rate of engagement by story type to determine what kind of content engaged users the most.

Because of the configurability of our system, we have introduced a number of parameters to tweak the new AF experience with generated content. For our first experiment, we started with some sane defaults, and it proved to be a success: users were favoriting more listings (+1.07%), favoriting more shops (+8.56%), and visiting more listings (+0.32%). We also noticed that users were more likely to engage with targeted content than organic content by a wide margin (Figure 4). This gave us confidence in our changes and encouraged further experimentation.

With the knowledge that generated content was valuable in the context of the AF, we set out to optimize the blend of content. We launched a 3-variant experiment where each variant had a different ratio of story types: 1) Listing-heavy: received significantly more listing recommendations, 2) Shop-heavy: received significantly more shop recommendations, and 3) Control: received an equal amount of all recommendation types. We found that both listing-heavy and shop-heavy outperformed our control. The shop-heavy variant resulted in lifts of +0.96% to item favorites, +14.94% to shop favorites, and +0.44% to listing views. Compare this to listing-heavy's results of +1.77%, +7.05%, and +0.26% respectively. While we saw significant engagement lifts across all of our metrics in both variants, shop-heavy resulted in the best results overall, and was launched to 100% of users.

6. CONCLUSION

In this paper, we presented an overview of the combined engineering and data science effort that led to a substantial improvement of the existing Etsy Activity Feed, which now acts as a platform to deliver any kind of content - ranging from recommendations to advertisements. From a usability perspective, the Activity Feed is now the homepage to tens of millions of users, and proves to be a pleasant experience, be it their first or 100th visit to Etsy. Users can continue to shape the content of their feed by choosing who to follow, while also receiving targeted content for discovering new items and trends. From an engineering standpoint, we have devised a scalable, modular, and configurable content delivery system that gives us tremendous control over the timing and type of general content that users see everyday. We have shared our framework for experimentation and demonstrated how it guided our decision making. The system has been deployed to 100% of our user base and we have seen a substantial increase in engagement metrics all across the board as a result.

7. ACKNOWLEDGMENTS

The authors would like to thank: Josh Attenberg and Rob Hall for providing insight and work on recommendation algorithms; James Lee and Vernon Thommeret for work on content balance, product requirements, and managing and directing the Activity Feed product development and launch; Junghoon Park and Rachel Nash for providing design input and a flexible home for our content.

8. REFERENCES

[1] D. Agarwal, B.-C. Chen, R. Gupta, J. Hartman, Q. He, A. Iyer, S. Kolar, Y. Ma, P. Shivaswamy, A. Singh, and L. Zhang. Activity ranking in linkedin feed. KDD '14, 2014.

[2] L. Backstrom and J. Leskovec. Supervised random walks: Predicting and recommending links in social networks. WSDM '11, 2011.

[3] D. M. Blei, A. Y. Ng, M. I. Jordan, and J. Lafferty. Latent dirichlet allocation. *JMLR*, 2003.

[4] M. Burke, C. Marlow, and T. Lento. Feed me: Motivating newcomer contribution in social network sites. CHI '09, 2009.

[5] P. Gupta, A. Goel, J. Lin, A. Sharma, D. Wang, and R. Zadeh. Wtf: The who to follow service at twitter. WWW '13, 2013.

[6] N. Halko, P. Martinsson, and J. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2), 2011.

[7] D. Hu, R. Hall, and J. Attenberg. Style in the long tail: Discovering unique interests with latent variable models in large scale social e-commerce. *KDD*, 2014.

[8] J. Kleinberg. Hubs, authorities, and communities. In *ACM Computer Survey*, 1999.

[9] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 2009.

[10] M. Zuckerberg, R. Sanghvi, A. Bosworth, C. Cox, A. Sittig, C. Hughes, K. Geminder, and D. Corson. Dynamically providing a news feed about a user of a social network, Feb. 23 2010. US Patent 7,669,123.