How Shopify Uses Recommender Systems to Empower Entrepreneurs

Chen Karako Follow
Jun 26, 2018 · 8 min read

Authors: Chen Karako **and** Dóra Jámbor

There is a good chance you have come across a “recommended for you” statement somewhere in our data-driven world. This may be while shopping on Amazon, hunting for new tracks on Spotify, looking to decide what restaurant to go to on Yelp, or browsing through your Facebook feed—ranking and recommender systems are an extremely important feature of our day-to-day interactions.

This is no different at Shopify, a cloud-based, multi-channel commerce platform that powers over 600,000 businesses of all sizes in approximately 175 countries. Our customers are merchants that use our platform to design, set up, and manage their stores across multiple sales channels, including web, mobile, social media, marketplaces, brick-and-mortar locations, and pop-up shops.

Shopify builds many different features in order to empower merchants throughout their entrepreneurial lifecycle. But with the diversity of merchant needs, and the variety of features that Shopify provides, it can quickly become difficult for people to filter out what’s relevant to them. We use recommender systems to suggest personalized insights, actions, tools and resources to our merchants that can help their

businesses succeed. Every choice a merchant makes has consequences for their business and having the right recommendation at the right time can make a big difference.

In this post, we'll describe how we design and implement our recommender system platform.

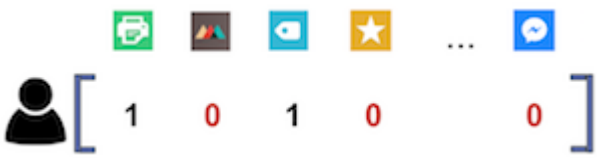
Methodology

Collaborative Filtering (CF) is a common technique to generate user recommendations for a set of items. For Shopify, users are merchants, and items are business insights, apps, themes, blog posts, and other resources and content that merchants can interact with. CF allows us to leverage past user-item interactions to predict the relevance of each item to a given user. This is based on the assumption that users with similar past behavior will show similar preferences for items in the future.

The first step of designing our recommender system is choosing the right representation for user preferences. One way to represent preferences is with user-item interactions, derived from implicit signals like the user's past purchases, installations, clicks, views, and so on. For example, in the Shopify App Store, we could use 1 to indicate an app installation and 0 to represent an unobserved interaction with the given app.

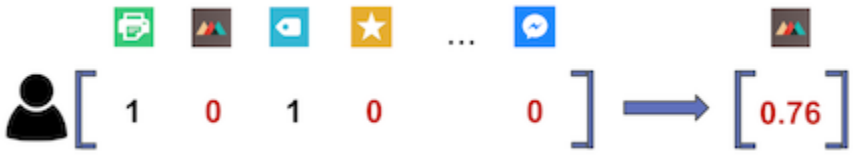
User-item interaction

These user-item interactions can be collected across all items, producing a user preference vector.



User preference vector

This user preference vector allows us to see the past behavior of a given user across a set of items. Our goal is now to predict the relevance of items that the user hasn't yet interacted with, denoted by the red 0s. A simple way of achieving our goal is to treat this as a binary classification problem. That is, based on a user's past item interactions, we want to estimate the probability that the user will find an item relevant.



User preference (left) and predicted relevance (right)

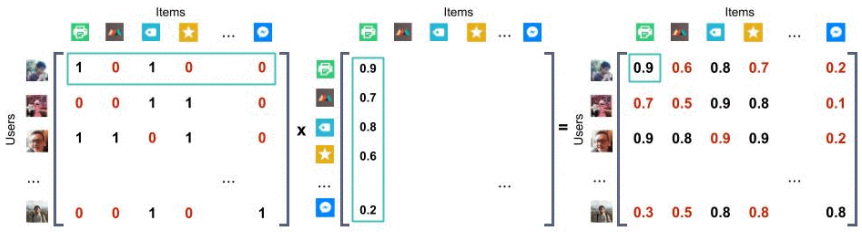
We do this binary classification by learning the relationship between the item itself and all other items. We first create a training matrix of all user-item interactions by stacking users' preference vectors. Each row in this matrix serves as an individual training example. Our goal is to reconstruct our training matrix in a way that predicts relevance for unobserved interactions.

There are a variety of machine learning methods that can achieve this task including linear models such as Sparse Linear Methods (SLIM), autoencoders, and matrix factorization. Despite the differences in how these models recover item relevance, they can all be used to reconstruct the original training matrix.

At Shopify, we often use linear models because of the benefits they offer in real-world applications. For the remainder of this post, we'll focus on these techniques.

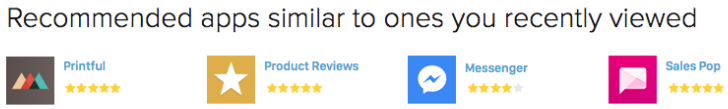
Linear methods like SLIM solve this optimization problem by directly learning an item-item similarity matrix. Each column in this item-item similarity matrix corresponds to an individual item's model coefficients.

We put these pieces together in the figure below. On the left, we have all user-item interactions, our training matrix. In the middle, we have the learned item-item similarity matrix where each column corresponds to a single item. Finally, on the right, we have the predicted relevance scores. The animation illustrates our earlier discussion of the prediction process.



User-item interactions (left), item-item similarity (middle), and predicted relevance (right)

To generate the final user recommendations, we take the items that the user has not yet interacted with, and sort their predicted scores (in red). The top scored items are then the most relevant items for the user, and can be shown as recommendations as seen below.



Personalized app recommendations on the Shopify App Store

Linear methods and this simple binary framework are commonly used in industry as they offer a number of desired features to serve personalized content to users. The binary aspect of the input signals and classification allows us to maintain simplicity in scaling a recommender system to new domains, while also offering flexibility with our model choice.

Scalability and parallelizability

As shown in the figure above, we train one model per item on all user-item interactions. While the training matrix is shared across all models, the models can be trained independently from one another. This allows us to run our model training in a task-parallel manner, while also reducing the time complexity of the training. Additionally, as the number of users and items grows, this parallel treatment favors the scalability of our models.

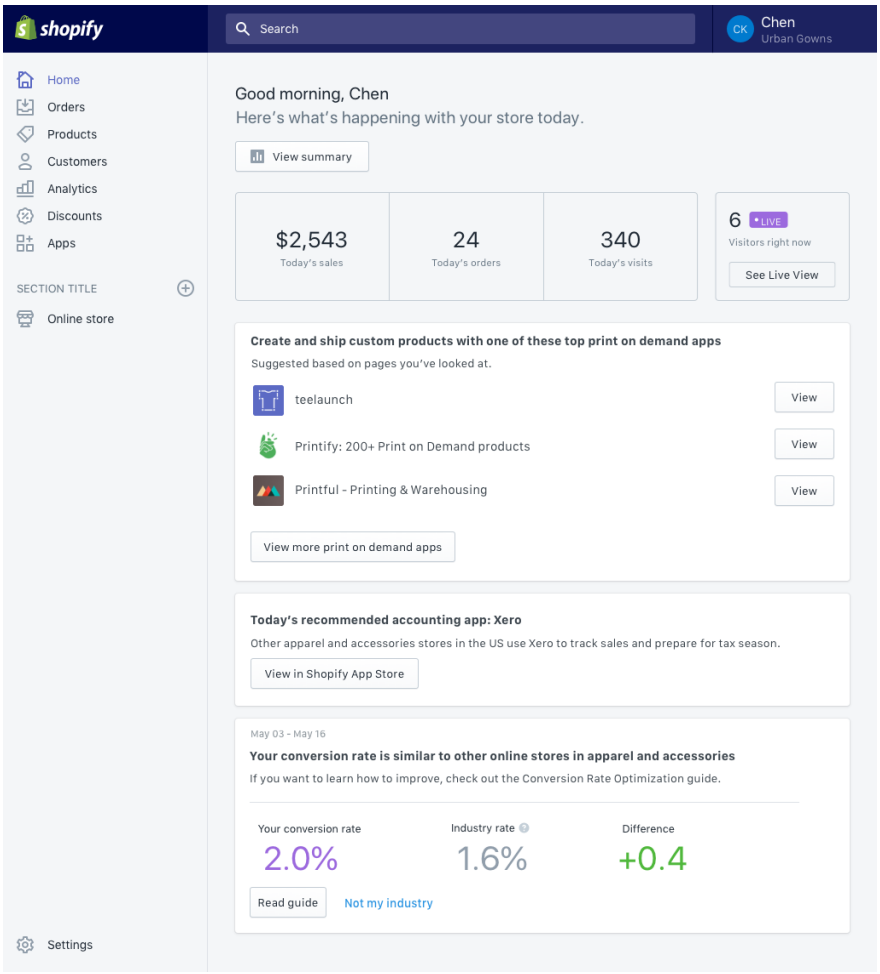
Interpretability

When building recommender systems, it's important that we can interpret a model and explain the recommendations. This is useful when developing, evaluating, and iterating on a model, but is also helpful when surfacing recommendations to users.

The item-item similarity matrix produced by the linear recommender provides a handy tool for interpretability. Each entry in this matrix corresponds to a model coefficient that reflects the learned relationship of two items. We can use this item-item similarity to derive which coefficients are responsible for a produced set of user recommendations.

Coefficients are especially helpful for recommenders that include other user features, in addition to the user-item interactions. For example, we can include merchant industry as a user feature in the model. In this case, the coefficient for a given item-user feature allows us to share with the user how their industry shaped the recommendations they see. Showing personalized explanations with recommendations is a great way of establishing trust with users.

For example, merchants' home feeds, shown below, contain personalized insights along with explanations for why those insights are relevant to them.



Shopify home feed, showing merchants how their business is doing, along with personalized insights.

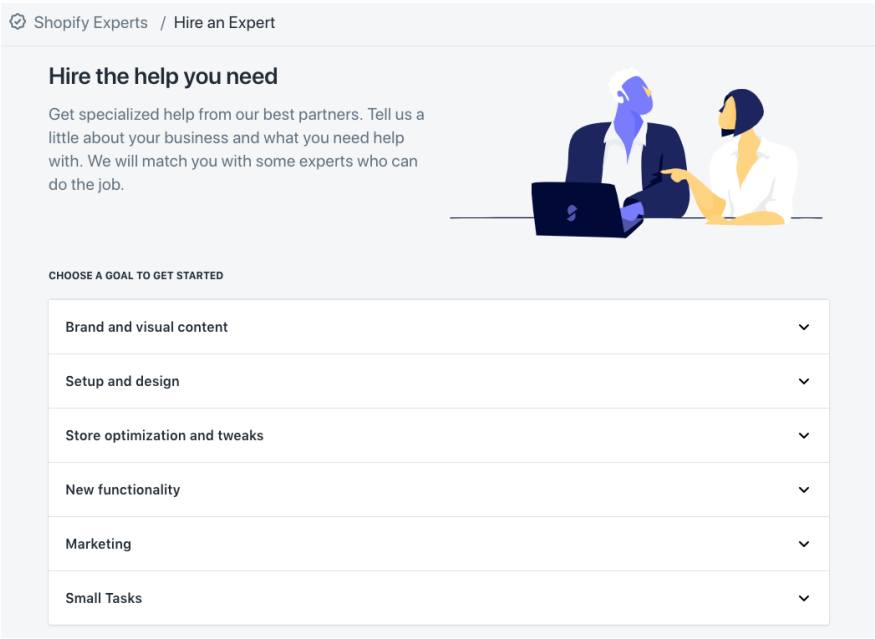
Extensibility

Beyond explanations, user features are also useful for enriching the model with additional user-specific signals such as shop industry, location, product types, target audience and so on. These can also help us tackle cold-start problems for new users or items, where we don't yet have much item interaction data. For example, using a user feature enriched model, a new merchant who has not yet interacted with any apps could now also benefit from personalized content in the App Store.

Performance

A recommender system must yield high quality results to be useful. Quality can be defined in various ways depending on the problem at hand. There are several recommender metrics to reflect different notions of quality like precision, diversity, novelty and serendipity. Precision can be used to measure the relevance of recommended items. However, if we solely optimize for precision, we might appeal to the majority of our users by simply recommending the most popular items to everyone, but would fail to capture subtleties of individual user preferences.

For example, the Shopify Services Marketplace, shown below, allows merchants to hire third-party experts to help with various aspects of their business.



Shopify Services Marketplace, where merchants can hire third-party experts

To maximize the chance of fruitful collaboration, we want to match merchants with experts who can help with their unique problems. On the other hand, we also want to ensure that our recommendations are diverse and fair to avoid scenarios in which a handful of experts get an overwhelming amount of merchant requests, preventing other experts from getting exposure. This is one example where precision alone isn't enough to evaluate the quality of our recommender system. Instead, quality metrics need to be carefully selected in order to reflect the key business metric that we hope to optimize.

While recommendations across various areas of Shopify optimize different quality metrics, they're ultimately all built with the goal of helping our merchants get the most out of our platform. Therefore, when developing a recommender system, we have to identify the metric, or proxy for that metric that allows us to determine whether the system is aligned with this goal.

Conclusion

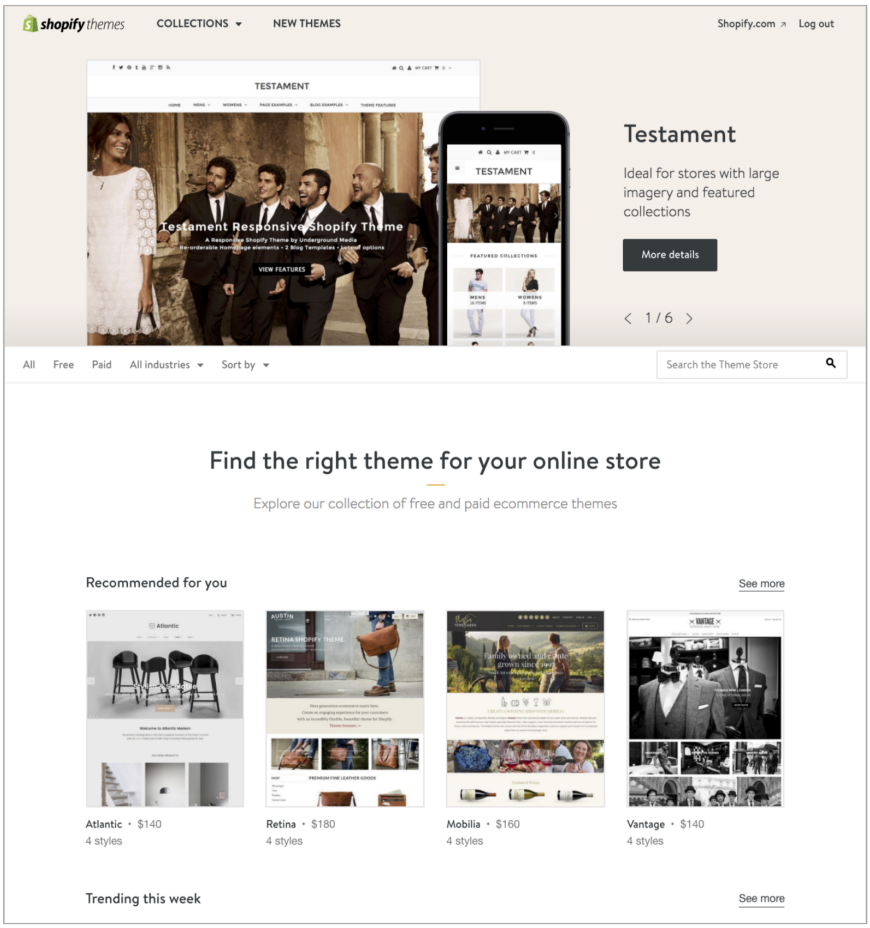
Having a simple and flexible base model reduces the effort needed for Discovery Algorithms team members to extend into new domains of Shopify. Instead, we can spend more time deepening our understanding of the merchant problems we are solving, refining key model elements, and experimenting with ways to extend the capabilities of the base model.

Moreover, having a framework of binary input signals and classification allows us to easily experiment with different models that enrich our recommendations beyond the capabilities of the linear model we presented above.

We applied this approach to provide recommendations to our merchants in a variety of contexts across Shopify. When we initially launched our recommendations through A/B tests, we observed the following results:

- Merchants receiving personalized app recommendations on the Shopify App Store had a 50% higher app install rate compared to those who didn't receive recommendations
- Merchants with a personalized home feed were up to 12% more likely to report that the content of their feed was useful, compared to those whose feeds were ranked by a non-personalized algorithm.

- Merchants who received personalized matches with experts in the Expert Marketplace had a higher response rate, and had overall increased collaboration between merchants and third-party experts.
- Merchants who received personalized theme recommendations on the Shopify Theme Store, seen below, were over 10% more likely to launch their online store, compared to those receiving non-personalized or no recommendations.



Shopify Theme Store, where merchants can select themes for their online store

We're always working on challenging new problems on the Shopify Data team. If you're passionate about leveraging data to help entrepreneurs, check out our open positions.