Bootstrapping Recommendations at Chrome Web Store

Zhen Qin, Honglei Zhuang, Rolf Jagerman, Xinyu Qian, Po Hu, Dan Chary Chen, Xuanhui Wang, Michael Bendersky, Marc Najork
Google Inc.
Mountain View, CA
{zhenqin,hlz,jagerman,xinyuqian,phu,lottie,xuanhui,bemike,najork}@google.com

ABSTRACT

Google Chrome, one of the world's most popular web browsers, features an extension framework allowing third-party developers to enhance Chrome's functionality. Chrome extensions are distributed through the Chrome Web Store (CWS), a Google-operated online marketplace. In this paper, we describe how we developed and deployed three recommender systems for discovering relevant extensions in CWS, namely non-personalized recommendations, related extension recommendations, and personalized recommendations. Unlike most existing papers that focus on novel algorithms, this paper focuses on sharing practical experiences when building large-scale recommender systems under various real-world constraints, such as privacy constraints, data sparsity and skewness issues, and product design choices (e.g., user interface). We show how these constraints make standard approaches difficult to succeed in practice. We share success stories that turn negative live metrics to positive ones, including: 1) how we use interpretable neural models to bootstrap the systems, help identifying pipeline issues, and pave the way for more advanced models; 2) a new item-item based algorithm for related recommendations that works under highly skewed data distributions; and 3) how the previous two techniques can help bootstrapping the personalized recommendations, which significantly reduces development cycles and bypasses various real-world difficulties. All the explorations in this work are verified in live traffic on millions of users. We believe that the findings in this paper can help practitioners to build better large-scale recommender systems.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; • **Computing methodologies** → **Neural networks**.

KEYWORDS

recommender systems, learning to rank, generalized additive models, text embedding

ACM Reference Format:
Zhen Qin, Honglei Zhuang, Rolf Jagerman, Xinyu Qian, Po Hu, Dan Chary Chen, Xuanhui Wang, Michael Bendersky, Marc Najork. 2021. Bootstrapping Recommendations at Chrome Web Store. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3447548.3467099

KDD '21, August 14–18, 2021, Virtual Event, Singapore.

ACM ISBN 978-1-4503-8332-5/21/08.
https://doi.org/10.1145/3447548.3467099

1 INTRODUCTION

Google Chrome, one of the premier web browsers, provides a framework that allows third parties to develop extensions that enhance the browser experience, e.g. play games, listen to music, edit photos, increase productivity, etc. Such extensions are distributed through the Chrome Web Store[1] (CWS), an online marketplace operated by Google. CWS provides functionalities that allows users to search for and serendipitously discover useful extensions. In this paper, we discuss how we developed three different CWS extension recommendation services from scratch, namely non-personalized recommendations, related extension recommendations, and personalized recommendations. We share techniques used and lessons learned in developing such industrial large-scale machine learning based recommender systems, with the goal to maximize user experience and minimize development cycles.

At first glance, it is tempting to solve these recommendation problems using machine learning approaches, which have been extensively studied recently (e.g. [7, 19]). However, to build a recommender system *from scratch*, the main challenge to utilize such an approach lies in obtaining the right training data for each specific application. Different from working with ready-to-use datasets in academic papers, the ideal data source for a standard machine learning formulation may not be readily available in practice. This situation could be due to several common reasons: 1) It is a new use case so there is no historical data for the particular application; 2) Even if there are production logs, the data may be biased because only results shown to users are collected; 3) Different data adjustment strategies are needed for different phases in a recommder system. For example, negative sampling is used to adjust the training data for the candidate generation phase, while this is usually not needed for the ranking phase [7].

On the other hand, there are several potential benefits in using machine learning approaches over heuristics [19]: 1) When done right, as we show in this paper, machine learning models can potentially work better than heuristics and "jump-start" the user experience. This is especially important for early-stage recommender systems, which need to attract and retain new users with high quality recommendations; 2) The sooner the first ML pipeline is ready and tested end-to-end, the earlier more advanced ML methods can be experimented with.

In this paper, we highlight the practical challenges and our counter strategies to overcome them. For non-personalized recommendations, we show how we design an interpretable neural

[1] https://chrome.google.com/webstore/

ranking model that bridges between hand-crafted heuristic approaches and full-fledged neural models. Interpretability helps with data debugging and feature analysis, thereby enabling us to quickly deploy the first versions of the training and inference pipelines. It paves the way for fast follow-up launches using more advanced machine learning techniques. For related recommendations, we show that it is non-trivial to use the user co-install data to train a machine learning model directly due to the popularity bias in the data. This also makes it harder to use co-install data for offline evaluation. We heavily employ data analysis during model development to understand the patterns. Based on this analysis, item-based collaborative filtering techniques [27] are adapted for this use case because they are easier to be adjusted. For personalized recommendations, we show that training a machine learning model based on user install history is challenging due to data sparsity and popularity bias. We propose a method that bootstraps with the previously developed related and non-personalized recommendations, and show it is effective in this case.

To summarize, the main contributions of this paper are as follows:

- We share our experience in developing three large-scale recommender systems from scratch, sharing both success stories and failures when handling various real-world constraints. To the best of our knowledge, few existing publications focus on this early stage of developing high-performance and large-scale recommender systems.
- Technically, we introduce how we design an interpretable neural ranking model that is both effective and easy to debug. We also introduce a new item-item based recommendation algorithm that mitigates highly skewed data and cold start problems.
- We show a strategy to effectively and efficiently bootstrap one recommendation use case using other use cases. This provides a perspective on a holistic treatment of different recommender modules within the same system, which is rarely explored in the literature.

The rest of the paper is organized as follows. In Section 2, we give an overview of the CWS platform and introduce the three recommendation problems. We then present each recommendation product specifically: non-personalized recommendation in Section 3, related recommendation in Section 4, personalized recommendation in Section 5, followed up by a discussion of next steps in Section 6. Related work is reviewed in Section 7. We conclude the paper in Section 8.

2 OVERVIEW OF CWS RECOMMENDATIONS

Chrome web store (CWS) provides a platform for millions of users to search, discover, and install hundred of thousands of extensions. Currently there are three recommendation modules in the store. *Non-personalized recommendations* for non-signed in users and *personalized recommendations* for signed-in users are served in the "Recommended For You" module with multiple recommended Chrome extensions on the CWS homepage, as shown in Figure 1. Though they are served at the same location, the problem setting and engineering architectures are very different, thus they are treated as two products in the paper (and within the development team). *Related recommendations* is the module shown when user clicks on the "Related" tab when they are at an extension detail page, as shown in Figure 2. We give more details of the product descriptions in their respective sections.

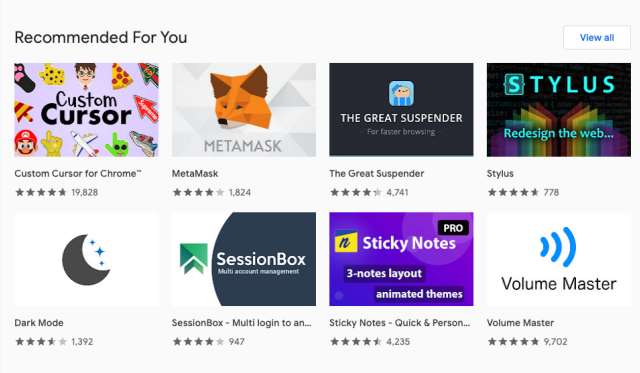


Figure 1: An illustration of the "Recommended For You" module on the CWS homepage. Non-personalized recommendations use it for non-signed in users and personalized recommendations use it for signed in users.

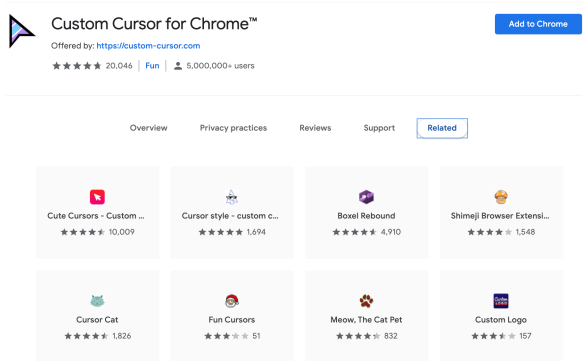


Figure 2: CWS related recommendations on the detail page of extension "Custom Cursor for Chrome".

When the team started working on the projects, non-personalized recommendations was a new use case. Personalized recommendations and related recommendations were supported by some legacy heuristics based systems. Those systems were strong quality-wise, but used expensive and hard-to-maintain components with high technical debts. The goal of the team was to completely replace them with modern architectures and algorithms for easy maintenance and extensibility. Thus, we did not re-use any components (including data, algorithm, and infrastructure) from the previous systems and these two cases can also be treated as new. However, one benefit is, we can *compare* with the previous systems in online A/B experiments, which we leverage to evaluate various algorithms. Our initial goal was to replace them with quality-neutral metrics, but we ended up significantly improving upon them by using techniques introduced in this paper. Next we describe each product in more detail.

3 NON-PERSONALIZED RECOMMENDATIONS

We describe how we built the non-personalized recommendations. The first iteration focuses on how we developed a novel interpretable neural generalized additive learning to rank model. This approach helps identify the data bias and mitigates training/serving discrepancy for the new use case, while not sacrificing much quality. We then briefly show how a more powerful deep model is deployed without much effort, given that the first iteration has sanity checked the entire pipeline of the desired setting.

3.1 Product description

The non-personalized recommendation product is designed to recommend Chrome extensions without using any user-specific information such as the extension install history. The only information from users are general context information, such as users' region and language settings from the browser. Ideally, the product is able to automatically recommend extensions that are popularly clicked and installed by users from a certain region and/or using the same language. The non-personalized recommendation product is primarily developed to serve the "Recommended For You" module at the top of the CWS homepage for users who are *not* signed-in (Figure 1), but the results can also benefit other products as described later.

3.2 Challenges

Since this is a new use case, the most substantial challenge is to set up the entire pipeline from end to end, including collecting training data, serving the predicted results, and building the evaluation metrics. Without an existing pipeline to adapt from, it would be extremely difficult to identify and pinpoint potential issues during the development process. The complexity of machine learning models makes troubleshooting even more challenging.

3.3 Bootstrapping with an interpretable model

Training data We utilize *anonymized* user logs on CWS as training data. The log consists of a set of user sessions. Each log entry contains all the extensions $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ shown to the user in that session, where each $\mathbf{x}_i = (x_{i1}, \ldots, x_{ik})$ is a k-dimensional feature vector representing an extension. An entry also captures which extensions were clicked/installed, represented by $\mathbf{Y} = \{y_1, \cdots, y_n\}$, where y_i indicates whether extension $\mathbf{x}_i$ was clicked/installed. We set the label $y_i = 2$ when the extension was installed, $y_i = 1$ when the extension was clicked, and $y_i = 0$ otherwise.

Another important signal is the context information of users. Each entry includes the region and language configurations of the user's browser, but excludes all other user-specific information. We represent the context information as a feature vector $\mathbf{q}$.

Interpretable model To facilitate the development, we opted to train interpretable models instead of black-box models as the first version of the product. Particularly, we train generalized additive models (GAMs). The model builds a sub-model for each feature separately, and takes the sum of all the sub-models' outputs as the final prediction. Particularly, the predicted score $\hat{y}$ is:

$$\hat{y} = F(\mathbf{x}) = \sum_j f_j(x_j) \tag{1}$$

where x_j is the j-th feature in the feature vector $\mathbf{x}$ and $f_j(\cdot)$ is the corresponding sub-model.

Although the performance may not be optimal compared with fully-fledged deep neural networks, such a model provides the transparency that one needs to set up a first-version machine learning model. Developers can visualize, examine or even edit each sub-model to understand whether a particular bug is due to the model or the data.

Traditional GAMs are not optimized for ranking, nor can they take context features such as users' regions and languages as input. Therefore, we propose and apply a neural ranking generalized additive model (Neural RankGAM) [43]. The model is instantiated by neural networks and is capable of optimizing ranking losses. Moreover, the model can be extended (Neural RankGAM+) to further include context features. Specifically, the model calculates the ranking score for each extension as:

$$\hat{y} = F(\mathbf{q}, \mathbf{x}) = \sum_{j=1}^{k} \Big(w_j(\mathbf{q}) f_j(x_j) \Big) \tag{2}$$

where $w_j(\cdot)$ maps the context feature vector $\mathbf{q}$ to a scalar weight for the j-th sub-model. A graphical illustration of Neural RankGAM+ can be found in Figure 3.

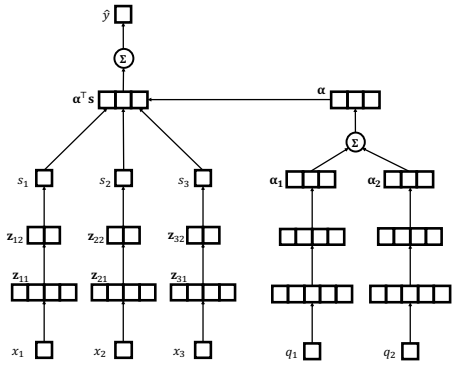
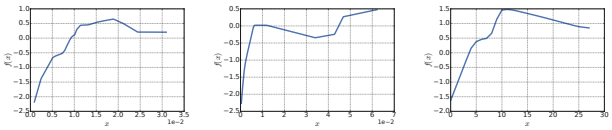


Figure 3: A graphical illustration of a context-present neural ranking GAM (Neural RankGAM+).

Troubleshooting examples We share some example cases during the development of the product to illustrate how the interpretable model is helpful in troubleshooting a complicated machine learning-based pipeline.

First, we always visualize all the features' sub-models to make sure they make sense (see Figure 4 for some examples). During the examination, we found some sub-models with abnormal shape. With further investigation, we were able to identify bugs from the training data generation pipeline, which resulted in corrupted feature values. Without the transparent Neural RankGAM, such bugs would not be obvious and can take much longer to be identified.



(a) Feature 7 (b) Feature 11 (c) Feature 15

Figure 4: Learned sub-model $f(x)$ of Neural RankGAM+ for selected features on CWS data set.

The second example occurred during the online deployment of the model. Although we obtained positive results in offline evaluation, the model did not deliver positive results in online experiments. By examining the sub-score of several extensions served in the recommended results, we found that there was a noticeable discrepancy between the training data distribution and the serving data distribution. The training data we collected largely contained the most popular extensions which were frequently shown on the CWS homepage, whereas the candidate extensions during serving could include many extensions that never appear in the training data. The feature values of such extensions often deviate so much from the distribution of the training data that makes the corresponding sub-model output an extreme score.

Based on this discovery, we further worked on aligning the distribution between data used in training and serving. There are several options, such as weighting training data or introducing confidence intervals on serving scores. For simplicity, we simply added a candidate filter in the serving pipeline. We calculate the frequency of candidate extensions in the training data and remove extensions that do not appear more frequently than a certain threshold. We only serve extensions that appeared frequently enough in the training data. The online performance turned positive with this filter applied. We plan to work on more sophisticated techniques for resolving this issue.

3.4 Experiments

Offline experiment We compare with other interpretable alternatives in the offline experiments, including Tree GAM [20], Tree RankGAM [18], as well as a variation of Neural RankGAM which optimizes regression loss (Neural GAM). We also compare both the context-absent version of Neural RankGAM and the context-present version of Neural RankGAM (denoted as Neural RankGAM+). Table 1 shows the performance.

As one can clearly see, our proposed Neural RankGAM is as competitive as tree-based counterpart. More importantly, the extended context-based Neural RankGAM+ can effectively leverage the context features and further improve the performance significantly.

Based on the results, we move on with both Neural RankGAM and Neural RankGAM+ models, where Neural RankGAM+ serve users with region/language configurations sufficiently frequent in the training data, and the context-absent Neural RankGAM serve users with less frequent region/language settings.

Online experiment In online experiments, we verify whether serving the non-personalized recommendation results in the "Recommended for You" block can actually help users without signing in. Table 2 shows the online A/B experiment performance for 2 weeks of the launched model. The results show that by adding this block, we can significantly improve user engagement metrics in terms of both clicks and installs on the entire homepage.

Table 1: Offline performance comparison for non-personalized recommendation. Results that are statistically significantly better ($\alpha = 0.01$ using a two-tailed t-test) than *Tree RankGAM* are marked with $^{\uparrow}$, and the best results are bolded.

Method	NDCG@1	NDCG@5	NDCG@10
Tree GAM	0.1974	0.3291	0.3672
Neural GAM	0.2009	0.3401	0.3860
Tree RankGAM	0.2016	0.3506	0.3927
Neural RankGAM	0.2035	0.3494	0.3893
Neural RankGAM+	**0.2443**$^{\uparrow}$	**0.3988**$^{\uparrow}$	**0.4284**$^{\uparrow}$

Table 2: Online relative performance on the CWS homepage compared with not showing the "Recommended For You" module for users who are not signed in. ↑ denotes a statistically significant increase ($\alpha = 0.01$ using a two-tailed t-test).

Model	Click Number	Install Number
Neural RankGAM+	+1.91%$^{\uparrow}$	+0.98%$^{\uparrow}$

3.5 Follow-up model development

By deploying the interpretable RankGAM model for bootstrapping non-personlized recommendations, we were able to set up the entire pipeline in the TensorFlow ecosystem [1, 22, 24]. Our next launch simply uses a more complex fully connected neural network, more sophisticated feature transformation [44] and more training data with significant gains (+3.08% clicks, +1.65% installs on the CWS homepage). The iteration is much faster considering that the Tensorflow pipeline has been stress-tested, and the engineers are allowed to largely focus on model quality improvements.

4 RELATED RECOMMENDATIONS

We introduce the development of Related extension recommendations. We describe the challenges and two failed attempts from which we learned. Then we introduce a new hybrid item-item based recommendation method that led to significant gains during online A/B experiments.

4.1 Product description

Each extension in CWS has its own detail page. It has a "Related" tab that recommends similar or complementary extensions (Figure 2). The related tab is one of the major discovery methods in CWS. It allows users to find extensions relevant to the owner of the detail page, which is called the *context extension* in this section.

4.2 Challenges

When the team started working on the project, there was a legacy heuristic-based system running in production. We will describe this system in more detail later. Despite the already deployed system,

activities on the extension detail page were not being logged due to logging privacy constraints. We only have anonymized user co-install data: given an anonymized user, we know which items they have installed. The main challenge we face here is how to build a related recommendation system using solely co-install data.

We believe that this is a common setting in practice and our experience can help practitioners. First, the *related* recommendations use case is popular and frequently used on many online websites, such as Pinterest [19]. While most papers on recommender systems are user-centric, related recommendations are item-centric. Second, the lack of actual logs from the extension detail page is a common problem if it is a new product or there are limitations on what can be logged, both from a privacy and engineering standpoint, as is the case with our system. Lastly, user co-install data is actually one of the most common data formats for recommender system research [27]. In what follows, we show how practitioners can leverage such data to build a related recommendation system.

4.3 The production baseline

As we mentioned, there was a legacy heuristic-based production system. It had reasonable quality – as we show below, some standard off-the-shelf recommendation algorithms perform substantially worse than this production system. However, the system used an internal Knowledge Graph (KG) based engine to match attributes of the extensions. The call latency to the KG service was causing severe performance issues, and the complex hand-crafted rules were difficult to evolve. Therefore, our plan was to completely replace the legacy system. Indeed, we no longer use any components from this system, so the development of related recommendations can be treated as a new use case built from scratch. Meanwhile, we were able to compare it with the legacy system in online A/B experiments to measure different methods.

4.4 First attempt: Pointwise mutual information

Our first attempt was to leverage the classic pointwise mutual information (PMI) [6]. Given co-install data, we can treat each user as a collection and calculate pairwise scores between two extensions e_1 and e_2 as follows:

$$PMI(e_1, e_2) = \frac{N \cdot \#(e_1, e_2)}{\#(e_1)\#(e_2)} \propto \frac{\#(e_1, e_2)}{\#(e_1)\#(e_2)} \tag{3}$$

where N is the total number of installs, $\#(e_i)$ is the number of times extension e_i is installed, and $\#(e_1, e_2)$ is the number of times e_1 and e_2 are co-installed by a user. During inference, given context extension e_1, we rank all candidate extensions by $PMI(e_1, \cdot)$ and serve the top results as related results. We note that $\#(e_1)$ can be ignored in implementation, and other variants such as normalizing the counts or adding logarithm will have the same results for this application.

We conducted online A/B experiments and the results are shown in Table 3. We can see the results are shockingly bad. This also shows the production baseline is a strong one to match.

Cold start problem and UI implications After closely examining the results, we found two major problems. First, PMI, which depends on statistical counting, does not work for the many tail extensions with very few installs. This is a well-known cold start problem for recommender system and the problem is more severe for related recommendations, since we are generating results for *every* extension. Second, as has been observed in the literature [17], PMI tends to find relevant but rare items. This could be useful in some applications, but when we look at Figure 2, we can see the user interface exposes the number of ratings of each extensions, which roughly correlates with popularity. When we show rare extensions without many ratings, users just tend to ignore them, regardless of relatedness. Thus, production implications such as user interface design should also be considered when developing real-world recommender systems, and popularity is a factor we need to consider in this use case, which PMI cannot satisfy.

Table 3: Online relative performance compared with the production baseline using PMI for related recommendations. ↓ denotes a statistically significant decrease ($\alpha = 0.01$ using a two-tailed t-test).

Model	Click Rate	Install Rate
PMI	-28.93%↓	-32.23%↓

4.5 Second attempt: A learning to rank formulation

Motivated by the success of non-personalized recommendations, our second attempt was to build a neural learning to rank model using co-install data. To obtain training examples, we use the following approach: given a user's co-install data, for example $\{e_1, e_2, e_3\}$, we randomly sample one extension, say e_1, to serve as the context/query. Then e_2 and e_3 are treated as positive examples, and we randomly sample some negative examples from the entire corpus. Now this is a standard learning to rank problem setting and different models can be trained on such data. The features we started with were extension ids. During inference, given a context extension and the trained model, we can rank the entire corpus and use the top results as the related extensions. It is not hard to show that this approach is equivalent to the neural collaborative filtering approach [14], though the context here is an extension, not a user.

4.5.1 Popularity bias. Manual inspection of initial results clearly showed the failure of the first trained model – virtually all context extensions have the *same* set of most popular related extensions, and we did not even bother testing it online. This is understandable since CWS extension install numbers possess a highly skewed distribution that is common in practice – most positive examples in the learning to rank setting are the few most popular extensions, so the model memorizes them. Though it might make some sense to recommend the most popular items for user-centric applications (in fact, many websites have popularity based recommendation modules), such results are disastrous to related recommendations since the most popular items will be completely unrelated to the context extension in consideration.

4.5.2 Importance weighting is hard to tune. After realizing the highly skewed data, we tried different methods including example weighting (e.g., downweight the popular items), more sophisticated

Table 4: Online relative performance compared with the production baseline using learning-to-rank (LTR) for related recommendations. ↓ denotes a statistically significant decrease ($\alpha = 0.01$ using a two-tailed t-test).

Model	Click Number	Install Number
LTR	-32.30%↓	-47.13%↓

negative sampling [36], and adding more generalizable features such as textual extension titles and descriptions. For each of these variants, we inspect them manually. Once we had a model that passed manual inspection, we moved to online A/B experiments. The results are shown in Table 4.

The results turned to be even worse than PMI. We found that as we started to manipulate the training data via example weighting, the model goes from one extreme of recommending popular items to another extreme of recommending lexically similar but unpopular items. The lesson is that it is very difficult to control the trade-off between popularity and relatedness on a real-world data using a pure learning-based approach. We needed more controllable methods to bootstrap the co-install data for related recommendations.

4.6 A new hybrid item-item recommendation method

We note that PMI is more controllable and has clear failure patterns. In this section, we propose a novel mixture model for the co-install data to overcome the popularity bias. We find that this method has a complementary pattern compared with PMI. Thus we combine them in a new hybrid approach that mitigates the drawbacks of each other.

4.6.1 Mixture model. Given an extension e_1, we would like to return a list of e_2's that are related to e_1. The absolute co-install count of $\#(e_1, e_2)$ is biased towards the popular extensions. We propose a generative mixture model to overcome this bias. The mixture model assumes that a co-install of e_2 given e_1 is generated by a mixture of two components: e_2 is related to e_1, and e_2 is randomly picked based on its popularity:

$$P_{\text{installed}}(e_2|e_1) = (1-\lambda) \cdot P_{\text{related}}(e_2|e_1) + \lambda \cdot P(e_2)$$

where λ is the mixture weight and set to 0.99 in our use case and

$$P(e_2) \propto \#(e_2).$$

We are interested in estimating $P_{\text{related}}(e_2|e_1)$. We can define a maximum likelihood problem using the co-install counts

$$L = \sum_{e_2} \#(e_1, e_2) \log \left(P_{\text{installed}}(e_2|e_1)\right).$$

This can be done by a standard Expectation-Maximization (EM) algorithm [8] as shown in [38], which is slow to converge. Fortunately, a fast exact method was discovered for this simple mixture model in [39]. It only needs to sort the extensions once and then perform a few linear scans of the sorted extensions to compute the exact $P_{\text{related}}(e_2|e_1)$ values.

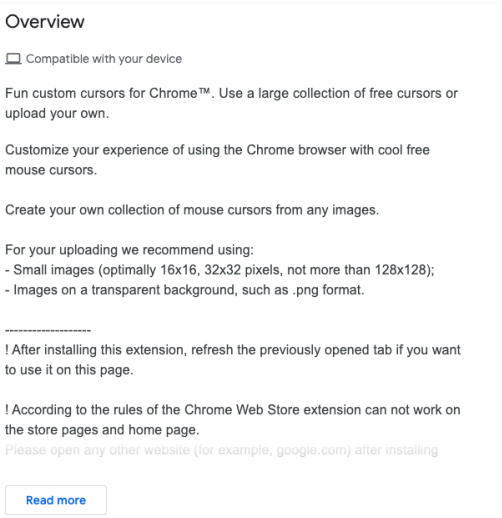


Figure 5: An illustration of the textual descriptions for the extension "Custom Cursor for Chrome".

4.6.2 Hybrid method. By inspecting the results, we found that the mixture model gives quite sensible extensions that tend to be popular. This is complementary to the patterns of PMI. We thus combine them together by weighted sum:

$$PMI(e_1, e_2) + w * P_{\text{related}}(e_2|e_1)$$

where w is set to 5.0 manually. We use this method for **head extensions**.

However, neither PMI nor the mixture model works for **tail extensions** that have very few installs, namely the cold-start problems. We propose to leverage the state-of-the-art pre-trained natural language processing models, such as BERT [9]. As shown in Figure 5, the extension developer would provide textual description of each extension, which we can feed into BERT to get its text embedding. We simply use the dot product of text embedding vectors as the base score followed by simple re-ranking based on popularity. Note that by leveraging pre-trained models, the method for tail extensions is completely unsupervised and requires minimal development cycles.

The results of the hybrid approach are shown in Table 5, running on millions of users for 2 weeks. We can see the results are very positive after we resolved all the major pain points of standard approaches. The hybrid approach completely replaced the previous system and is currently fully deployed to all users.

Table 5: Online relative performance compared with the production baseline using the hybrid approach for related recommendations. ↑ denotes a statistically significant increase ($\alpha = 0.01$ using a two-tailed t-test).

Model	Click Number	Install Number
LTR	+6.99%↑	+4.69%↑

5 PERSONALIZED RECOMMENDATIONS

We discuss how we built the personalized recommendations service at CWS. We explain why practical constraints make standard approaches difficult for this use case. We then introduce how we leverage non-personalized recommendations and related recommendations to help bootstrapping this use case with minimal development effort.

5.1 Product description

The same as non-personalized recommendations, the personalized recommendation product is developed to serve the "Recommended For You" module at the top of CWS homepage for users (Figure 1) who are *signed-in*, so we generate the recommendations based on each user's historical behaviors. Though served at the same place, personalized recommendations and non-personalized recommendations are two different products with substantial differences in terms of privacy requirements, data storage, and methodologies.

5.2 Challenges

Personalized recommendations is the standard research problem for recommender systems, especially in academic research. There are two common approaches in the recent literature: 1) Log user information and activities (such as clicks) in actual user sessions, and train machine learning models using these logs [7, 25]; 2) Given a user's installed extensions with associated install time stamps, build a predictive model under the sequential recommender system framework [21, 29, 30, 32, 33].

Option 1) can be time-consuming to realize nowadays due to user privacy and the non-trivial engineering efforts needed to properly store sensitive private data. Option 2) is a better choice since it only requires offline collection of users' installed extensions. This dataset is similar but slightly different from the one used in related recommendations in that each user is associated with an identifier. Otherwise we could not serve personalized recommendations. The temporal extent of each user's install history is shorter than that used in related recommendations due to privacy requirements.

5.3 The production baseline

The legacy system, which was based on heuristic rules and Knowledge Graph attributes similar to the legacy system for related recommendations in Section 4.3, had been running in production for a long time and delivered robust personalized recommendations. We do not use any components from that system but we were able to compare with it in online A/B experiments.

5.4 Attempt: A predictive learning to rank formulation

Similar to the approach introduced in Section 4.5, we started building a learning to rank model. Given a user's install history, for example $\{e_1, e_2, e_3\}$, ordered in time, we select a break point and formulate this as a prediction problem, say $\{e_1, e_2\} \rightarrow e_3$, we can treat $\{e_1, e_2\}$ as the context/query and e_3 as the positive example. We do negative sampling from the entire corpus. The features we started with were extension ids, textual descriptions, etc. During inference, given each user's installed history, we can rank the entire corpus and use the top results as the personalized recommendations.

However, similar to the learning to rank effort for related recommendations, this attempt is compromised by skewed and sparse data. The problem is more severe since the time span is even shorter than that used in related recommendations, and the data becomes even sparser when we split each sequence to past and future. We also tested with techniques such as example weighting and more advanced negative sampling techniques but the results were not satisfying.

5.5 A bootstrapping approach

Among the three projects introduced in this paper, the personalized recommendation project experienced the slowest progress due to its challenging setting both methodology- and process-wise. After we had successfully deployed highly-performant non-personalized and related recommendations services, we wondered whether we could leverage these services to bootstrap personalized recommendations and moreover to provide users a coherent CWS experience.

As it turns out, we can indeed, by using a simple yet effective approach. For a user with install history $\{e_1, e_2, e_3\}$, we retrieve the related results for e_1, e_2, and e_3 (as described in Section 4). We also retrieve the non-personalized recommendations based only on the user's browser setting (as described in in Section 3) as a 4th set for this example. We find the non-personalized recommendations are useful to improve diversity, especially when a user only installed few extensions. These extensions are ranked by simple heuristics using levels: At each level k (starting from 1), we pick the k-th results from the 4 sets, ordered them by popularity, and then place them at positions from $4k-3$ to $4k$. Such simple strategy further improves diversity. The 2 week online A/B results are shown in Table 6.

Table 6: Online relative performance compared with the production baseline leveraging non-personalized and related recommendations for personalized recommendations. "In module" means metrics within the *recommended for you* module, and "in homepage" means the CWS homepage. $\uparrow$ denotes a statistically significant increase ($\alpha = 0.01$ using a two-tailed t-test).

Click Number in module	Install Number in module	Click Number in Homepage	Install Number in Homepage
+9.92%$^{\uparrow}$	+10.50%$^{\uparrow}$	+2.83%$^{\uparrow}$	+3.08%$^{\uparrow}$

The performance improvements are very significant. We believe the strong results are due to leveraging the highly-performant related and context-based non-personalized recommendations. In fact, this strategy may be appealing from a user's perspective: after a user has installed e_1, they may not visit the detail page of e_1 often and see its related results. Similarly for non-personalized recommendations, when a user is logged in, they will not see the non-personalized recommendations. By bootstrapping personalized recommendations with the other two use cases, we are able to promote those high quality personalized results to the user's homepage. Interestingly, though the development of personalized

recommendations took the longest development time among the three services, the actual implementation of the working system took very short amount of time by leveraging established systems. The described system has been deployed and is serving all personalized recommendations in CWS.

Based on the lessons we learned, we encourage practitioners to think of the recommendation modules in a website as a coherent set, which can be useful for bootstrapping new use cases. This can provide users with a better overall experience in general, as demonstrated in this work. To the best of our knowledge, discussions of such topics are rare in the literature.

6 DISCUSSION AND NEXT STEPS

In this work, we focused on showing several techniques and strategies to build three large-scale recommendation products in CWS from scratch. We believe that our learned lessons and novel strategies to mitigate real-world constraints will be useful for practitioners who seek to "jump-start" user experience in their recommendation systems. The proposed techniques are also easily extendable, and we propose a few follow-up steps in particular:

- For non-personalized recommendations, as we mentioned, we are exploring more advanced machine learning techniques, such as more expressive models [9, 26], result diversification [35], and unbiased learning to rank [16, 42] to account for potential biases in logged user interactions.
- For related recommendations, we are working on the logging of actual session data while retaining maximal user privacy, and plan to build learning based models to further improve the current system. Fine-tuning BERT with such data to further improve performance for tail extensions is also plausible. On the other hand, we are still interested in a unified machine learning based approach where we can better balance the trade-off between popularity and relatedness.
- Similar to related recommendations, we are working on logging user activity data under privacy constraints, and we are researching unified models that work on highly skewed real-world data. On the other hand, any improvements from the other two products may improve personalized recommendations. We are also exploring other ways to better coordinate all recommendation use cases in CWS, and even interact with other Google products, such as web search.

7 RELATED WORK

Many case studies of industrial recommendation systems describe final systems, but they do not describe how one might build the system incrementally, especially the first iterations. A lot of work focuses on developing advanced machine learning models to further improve the quality of existing products, assuming functional pipelines and appropriate training data (e.g., from an existing deployed system) that are results of non-trivial engineering efforts and data accumulation.

Real-world recommender systems have been described for image search [15], video discovery on YouTube [7], movies on Netflix [11], and private documents in Google Drive [5]. Covington et al. [7] discuss work on improving YouTube's recommendation system, focusing on the implementation of the deep neural networks. Zhao et al. [41] use an advanced multi-task deep architecture to recommend what video to watch next on YouTube. Beutel et al. [2] design a novel algorithm to better use context information for Youtube Recommendations. Haldar et al. [13] give an introduction at how deep learning enabled them to significantly improve Airbnb search. Grbovic et al. [12] focus on building user and item embeddings for Airbnb search and recommendations. Yin et al. [37] describe ranking functions, semantic matching features, and query rewriting components for Yahoo search. Eide et al. [10] discuss the use of multi-armed bandits as a high-level reranker on top of other recommendations at the popular Norwegian website FINN.no. Xu et al. [34] describe how to use knowlegde distillation to better utilize unseen features during inference time at Taobao recommendations. Zhao et al. [40] study deep reinforcement learning on page-wise recommendations at JD.com.

Few existing work addresses real-world constraints that make the first iterations of machine learned recommender systems difficult and possibly time-consuming, such as privacy constraints, difficulties in debugging ML pipelines, and the lack of appropriate training data. Paleyes et al. [23] survey challenges at every step of the ML deployment workflow due to practical considerations of deploying ML in production. Our work focuses on bootstrapping large-scale recommender systems, proposing strategies to ameliorate real-world constraints, resulting in highly-functional systems that work significantly better than heuristics that are usually used for bootstrapping [19] with short development time. We leverage pre-trained NLP models [4, 9] and bootstrap one module with other modules, treating the entire Chrome Web Store as a coherent space. Such strategies are not well explored in the literature.

Most existing recommender system work focuses on user-centric recommendations. For *Related* recommendations, early work designs similarity functions based on domain expertise [31]. Brovman et al. [3] use coview data logged from the same browsing sessions in eBay. Schnabel et al. [28] focus on debiasing user feedback, assuming the existence of interaction logs and an extra annotated dataset. Liu et al. [19] cover the gradual improvement of the Pinterest "Related Pins" recommendation system. Besides the different use case and real-world constraints, Liu et al. [19] leverage user generated boards that consist of posts for pattern discovery. Such explicit collections are different from our implicit user install history, which tends to be much noisier and biased.

8 CONCLUSION

In this work, we showed how we built three large-scale recommendation products in Chrome Web Store from scratch. We focus on bootstrapping strategies for early iterations, proposing several techniques and strategies that outperform strong baselines and reduce development or debugging time. Our main findings are: 1) For early model development, interpretable models such as GAMs are helpful as they can aid troubleshooting, help find bugs, and discover data distribution discrepancies; 2) Popularity bias, especially for related recommendations, should be considered for model development, particularly when the user interface design exposes popularity information; 3) By re-using recommendation modules we can more efficiently bootstrap new use cases and provide users with a better experience. We believe that our learned lessons and

novel strategies are useful for practitioners to mitigate various real-world constraints and to rapidly and cost-efficiently deliver strong recommendation experiences to their users.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *USENIX Symposium on Operating Systems Design and Implementation*. 265–283.

[2] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H Chi. 2018. Latent cross: Making use of context in recurrent recommender systems. In *Proceedings of the 11th ACM International Conference on Web Search and Data Mining*. 46–54.

[3] Yuri M Brovman, Marie Jacob, Natraj Srinivasan, Stephen Neola, Daniel Galron, Ryan Snyder, and Paul Wang. 2016. Optimizing similar item recommendations in a semi-structured marketplace to maximize conversion. In *ACM Conference on Recommender Systems*. 199–202.

[4] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Céspedes, Steve Yuan, Chris Tar, et al. 2018. Universal sentence encoder. *arXiv preprint arXiv:1803.11175* (2018).

[5] Suming J Chen, Zhen Qin, Zac Wilson, Brian Calaci, Michael Rose, Ryan Evans, Sean Abraham, Donald Metzler, Sandeep Tata, and Michael Colagrosso. 2020. Improving Recommendation Quality in Google Drive. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2900–2908.

[6] Kenneth Church and Patrick Hanks. 1990. Word association norms, mutual information, and lexicography. *Computational linguistics* 16, 1 (1990), 22–29.

[7] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *ACM Conference on Recommender Systems*. 191–198.

[8] A. P. Dempster, N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B* 39 (1977), 1–38.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[10] Simen Eide and Ning Zhou. 2018. Deep neural network marketplace recommenders in online experiments. In *ACM Conference on Recommender Systems*. 387–391.

[11] Carlos A Gomez-Uribe and Neil Hunt. 2015. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems* 6, 4 (2015), 1–19.

[12] Mihajlo Grbovic and Haibin Cheng. 2018. Real-time personalization using embeddings for search ranking at airbnb. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 311–320.

[13] Malay Haldar, Mustafa Abdool, Prashant Ramanathan, Tao Xu, Shulin Yang, Huizhong Duan, Qing Zhang, Nick Barrow-Williams, Bradley C Turnbull, Brendan M Collins, et al. 2019. Applying deep learning to Airbnb search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1927–1935.

[14] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *The Web Conference*. 173–182.

[15] Yushi Jing and Shumeet Baluja. 2008. Pagerank for product image search. In *The Web Conference*. 307–316.

[16] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. 2017. Unbiased learning-to-rank with biased feedback. In *ACM International Conference on Web Search and Data Mining*. 781–789.

[17] Marius Kaminskas and Derek Bridge. 2014. Measuring surprise in recommender systems. In *Workshop on Recommender Systems Evaluation: Dimensions and Design*.

[18] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *International Conference on Neural Information Processing Systems*. 3149–3157.

[19] David C Liu, Stephanie Rogers, Raymond Shiau, Dmitry Kislyuk, Kevin C Ma, Zhigang Zhong, Jenny Liu, and Yushi Jing. 2017. Related pins at Pinterest: The evolution of a real-world recommender system. In *The Web Conference*. 583–592.

[20] Yin Lou, Rich Caruana, and Johannes Gehrke. 2012. Intelligible models for classification and regression. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 150–158.

[21] Yifei Ma, Balakrishnan Narayanaswamy, Haibin Lin, and Hao Ding. 2020. Temporal-Contextual Recommendation in Real-Time. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2291–2299.

[22] Christopher Olston, Noah Fiedel, Kiril Gorovoy, Jeremiah Harmsen, Li Lao, Fangwei Li, Vinu Rajashekhar, Sukriti Ramesh, and Jordan Soyke. 2017. Tensorflow-serving: Flexible, high-performance ml serving. *arXiv preprint arXiv:1712.06139* (2017).

[23] Andrei Paleyes, Raoul-Gabriel Urma, and Neil D Lawrence. 2020. Challenges in deploying machine learning: a survey of case studies. *arXiv preprint arXiv:2011.09926* (2020).

[24] Rama Kumar Pasumarthi, Sebastian Bruch, Xuanhui Wang, Cheng Li, Michael Bendersky, Marc Najork, Jan Pfeifer, Nadav Golbandi, Rohan Anil, and Stephan Wolf. 2019. TF-Ranking: Scalable tensorflow library for learning-to-rank. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2970–2978.

[25] Changhua Pei, Yi Zhang, Yongfeng Zhang, Fei Sun, Xiao Lin, Hanxiao Sun, Jian Wu, Peng Jiang, Junfeng Ge, Wenwu Ou, et al. 2019. Personalized re-ranking for recommendation. In *ACM Conference on Recommender Systems*. 3–11.

[26] Zhen Qin, Le Yan, Honglei Zhuang, Yi Tay, Rama Kumar Pasumarthi, Xuanhui Wang, Michael Bendersky, and Marc Najork. 2021. Are Neural Rankers still Outperformed by Gradient Boosted Decision Trees?. In *International Conference on Learning Representations*.

[27] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-Based Collaborative Filtering Recommendation Algorithms. In *The Web Conference*. 285–295.

[28] Tobias Schnabel and Paul N Bennett. 2020. Debiasing Item-to-Item Recommendations With Small Annotated Datasets. In *ACM Conference on Recommender Systems*. 73–81.

[29] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. In *ACM International Conference on Information and Knowledge Management*. 1441–1450.

[30] Jiaxi Tang, Francois Belletti, Sagar Jain, Minmin Chen, Alex Beutel, Can Xu, and Ed H. Chi. 2019. Towards neural mixture recommender for long range dependent user sequences. In *The Web Conference*. 1782–1793.

[31] Charlie Wang, Arpita Agrawal, Xiaojun Li, Tanima Makkad, Ejaz Veljee, Ole Mengshoel, and Alvin Jude. 2017. Content-based top-n recommendations with perceived similarity. In *IEEE International Conference on Systems, Man, and Cybernetics*. 1052–1057.

[32] Shoujin Wang, Liang Hu, Yan Wang, Longbing Cao, Quan Z Sheng, and Mehmet Orgun. 2019. Sequential recommender systems: challenges, progress and prospects. *arXiv preprint arXiv:2001.04830* (2019).

[33] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. 2017. Recurrent recommender networks. In *ACM International Conference on Web Search and Data Mining*. 495–503.

[34] Chen Xu, Quan Li, Junfeng Ge, Jinyang Gao, Xiaoyong Yang, Changhua Pei, Fei Sun, Jian Wu, Hanxiao Sun, and Wenwu Ou. 2020. Privileged Features Distillation at Taobao Recommendations. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2590–2598.

[35] Le Yan, Zhen Qin, Rama Kumar Pasumarthi, Xuanhui Wang, and Mike Bendersky. 2021. Diversification-Aware Learning to Rank using Distributed Representation. In *The Web Conference*.

[36] Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee Kumthekar, Zhe Zhao, Li Wei, and Ed Chi. 2019. Sampling-bias-corrected neural modeling for large corpus item recommendations. In *ACM Conference on Recommender Systems*. 269–277.

[37] Dawei Yin, Yuening Hu, Jiliang Tang, Tim Daly, Mianwei Zhou, Hua Ouyang, Jianhui Chen, Changsung Kang, Hongbo Deng, Chikashi Nobata, et al. 2016. Ranking relevance in yahoo search. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 323–332.

[38] Chengxiang Zhai and John Lafferty. 2001. Model-Based Feedback in the Language Modeling Approach to Information Retrieval. In *International Conference on Information and Knowledge Management*. 403–410.

[39] Yi Zhang and Wei Xu. 2007. Fast Exact Maximum Likelihood Estimation for Mixture of Language Models. In *ACM SIGIR Conference on Research and Development in Information Retrieval*. 865–866.

[40] Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2018. Deep reinforcement learning for page-wise recommendations. In *ACM Conference on Recommender Systems*. 95–103.

[41] Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumthekar, Maheswaran Sathiamoorthy, Xinyang Yi, and Ed Chi. 2019. Recommending what video to watch next: a multitask ranking system. In *Proceedings of the 13th ACM Conference on Recommender Systems*. 43–51.

[42] Honglei Zhuang, Zhen Qin, Xuanhui Wang, Mike Bendersky, Xinyu Qian, Po Hu, and Chary Chen. 2021. Cross-Positional Attention for Debiasing Clicks. In *The Web Conference*.

[43] Honglei Zhuang, Xuanhui Wang, Mike Bendersky, Alexander Grushetsky, Yonghui Wu, Petr Mitrichev, Ethan Sterling, Nathan Bell, Walker Ravina, and Hai Qian. 2021. Interpretable Ranking with Generalized Additive Models. In *ACM International Conference on Web Search and Data Mining*.

[44] Honglei Zhuang, Xuanhui Wang, Michael Bendersky, and Marc Najork. 2020. Feature transformation for neural ranking models. In *ACM SIGIR Conference on Research and Development in Information Retrieval*. 1649–1652.