Attribute-based Propensity for Unbiased Learning in Recommender Systems: Algorithm and Case Studies

Zhen Qin, Suming J. Chen, Donald Metzler, Yongwoo Noh, Jingzheng Qin, Xuanhui Wang
Google LLC, Mountain View, CA, USA
{zhenqin,suming,metzler,ynoh,jzq,xuanhui}@google.com

ABSTRACT

Many modern recommender systems train their models based on a large amount of implicit user feedback data. Due to the inherent bias in this data (e.g., position bias), learning from it directly can lead to suboptimal models. Recently, unbiased learning was proposed to address such problems by leveraging counterfactual techniques like inverse propensity weighting (IPW). In these methods, propensity scores estimation is usually limited to item's display position in a single user interface (UI).

In this paper, we generalize the traditional position bias model to an attribute-based propensity framework. Our methods estimate propensity scores based on offline data and allow propensity estimation across a broad range of implicit feedback scenarios, e.g., feedback beyond recommender system UI. We demonstrate this by applying this framework to three real-world large-scale recommender systems in Google Drive that serve millions of users. For each system, we conduct both offline and online evaluation. Our results show that the proposed framework is able to significantly improve upon strong production baselines across a diverse range of recommendation item types (documents, people-document pairs, and queries), UI layouts (horizontal, vertical, and grid layouts), and underlying learning algorithms (gradient boosted decision trees and neural networks), all without the need to intervene and degrade the user experience. The proposed models have been deployed in the production systems with ease since no serving infrastructure change is needed.

CCS CONCEPTS

• **Information systems** → **Information retrieval**; **Recommender systems**;

KEYWORDS

recommender system, implicit feedback, unbiased learning

ACM Reference Format:
Zhen Qin, Suming J. Chen, Donald Metzler, Yongwoo Noh, Jingzheng Qin, Xuanhui Wang. 2020. Attribute-based Propensity for Unbiased Learning in Recommender Systems: Algorithm and Case Studies. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3394486.3403285

KDD '20, August 23–27, 2020, Virtual Event, CA, USA

ACM ISBN 978-1-4503-7998-4/20/08.
https://doi.org/10.1145/3394486.3403285

1 INTRODUCTION

While traditional recommender system research heavily depends on explicit feedback (such as user ratings of movies [8, 37]), many modern recommender systems [20, 27, 38, 40, 44, 45] use implicit feedback, such as clicks, purchases, and dwell time, for training machine-learned recommendation models. There are several clear advantages to using implicit feedback, including: 1) it is abundant and inexpensive to collect, which is especially important nowadays since modern deep learning-based approaches [39] are generally data-hungry; and 2) users may be reluctant to explicitly rate items, or in some cases the UI itself may not even provide a means to collect explicit feedback from users.

However, implicit feedback is usually biased. Position bias is a well-known and often-studied type of bias [15, 19]. This bias focuses on the fact that users tend to react more favorably to the items at more visible positions [43]. Another type of bias is the so-called "previous-model" bias [27], where the model that is currently deployed dramatically impacts the training examples produced for future models. Directly using such biased feedback to train models can lead to suboptimal results.

Recently, unbiased learning was proposed to train unbiased models using counterfactual inference. A commonly used approach, that is based on importance sampling theory, is inverse propensity weighting (IPW) [30, 33]. During model training, examples are assigned different weights to account for the bias during feedback collection. Intuitively, items that have a lower propensity to receive feedback (e.g., those shown at a less visible position in the UI) should be assigned higher weights.

How to estimate the propensity weights is a critical task for unbiased learning. Based on the position bias model [29], online randomization experiments were proposed [24, 33, 34]. Such an approach unavoidably degrades the user experience [34], even when more mild interventions such as swapping adjacent pairs of items is done [24]. In practice, collecting data from online interventions is also typically more complex and requires more human effort than collecting implicit feedback signals.

Intervention harvesting [6, 16] was proposed recently for search ranking problems to exploit clicks from different ranking models. Such methods require multiple ranking models being deployed and can only be applied to feedback data collected directly from a single UI. Some work [7, 21] estimate the propensity weights and models jointly from click logs without any intervention, but did not consider bias attributes beyond result positions in a single UI. All these work focus on search, where a vertically listed documents is shown given a query, so modeling bias in a single UI might be sufficient. Real-world recommender systems, on the other hand, typically have more diverse UI designs (e.g., vertical vs grid), potentially on different platforms (e.g., mobile vs web).

More importantly, implicit feedback that comes from sources other than the recommendation UI itself is quite common and valuable. For example, e-commerce websites such as Amazon allows users to provide feedback on items outside the recommendation UI by searching and browsing on various product pages. An example of Google Drive's recommendation module, called "Quick Access" is shown in Fig. 1. Besides the recommendation UI, users have the flexibility to navigate through the website by simply browsing their organized documents or using the search functionality. This type of indirect or auxiliary feedback can be quite useful (e.g., it may help mitigate "previous-model" bias by leveraging a users' spontaneous exploration of the item space).

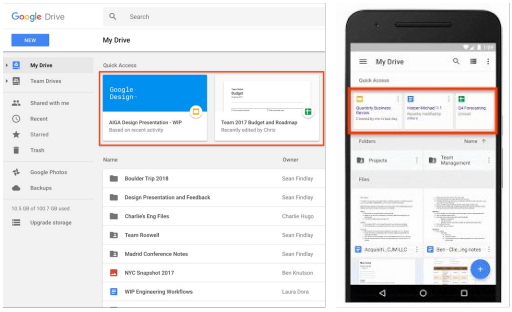

Figure 1: An illustration of Google Drive's Quick Access web (left) and mobile (right) versions from [1, 2], which recommend relevant documents to users based on the current context. Besides the recommendation module shown in red rectangle, users have various ways of accessing their documents and providing implicit feedback, including browsing their documents below the recommendation UI, using search, or using the navigation bar (left part of the web UI).

In this work, we focus on recommender systems and introduce an attribute-based propensity model for unbiased learning in this setting. We note that treating different factors, including positions, platforms (e.g. mobile vs web), and feedback sources (recommendation UI vs non-recommendation-UI) as "attributes" allows us to perform unified *offline* propensity weight estimation. Given the learned propensity weights, many learning algorithms can easily take advantage of them using existing model training APIs (e.g. Tensorflow [5] natively supports example weights), so the approach does not require any changes to how the resulting models are served online, which makes the approach highly practical. In addition, our estimated propensity weights allows for the design of unbiased offline evaluation metrics that we show that are better correlated with online metrics.

In summary, our contributions are as follows:

- We propose a general attribute-based propensity framework for unbiased learning in recommender systems that can be estimated offline. In addition to generalizing existing position bias models, the framework also generalizes across different UI layouts and can naturally incorporate a wide range of implicit feedback beyond that gathered in the recommendation UI itself.
- We conduct case studies on three large-scale real-world recommendation systems with different item types, layouts, and underlying modeling algorithms. We demonstrate significant benefits of our proposed framework. The proposed models have been deployed in production for an extended amount of time.
- We verify that unbiased offline metrics using these learned propensity weights are a better proxy than traditional metrics for online performance across all our use cases.

The rest of this paper is organized as follows. We review related work in Section 2 and then give a brief review of unbiased learning in Section 3. Our attribute-based propensity model and use cases are presented in Section 4 and Section 5. In Section 6, we present both offline and online evaluations of our proposed models. Finally, we conclude the paper in Section 7.

2 RELATED WORK

Recommender system research has primarily focused on explicit feedback, such as movie ratings [8, 25]. On the other hand, implicit feedback is getting more and more attention [20] in real-world large scale recommender systems, such as using clicks for image recommendation [27, 41], installations for mobile apps recommendation [13], clicks and watch time for video recommendation [11, 45], clicks and purchases for product recommendation in e-commerce websites [38, 44], and click and dwell time for news recommendation [26, 40]. Implicit feedback is abundant and easy to collect, which is especially important for data-hungry methods such as the recently popular deep learning approaches that can generalize traditional matrix factorization methods [18]. A thorough survey can be found at [42].

However, implicit feedback is known to be biased. For example, users tend to react differently to items placed at different positions [43]. This problem has drawn more attention in the information retrieval (IR) community recently [24, 33]. In IR applications, such as web search, results are often displayed to the user as a vertical list of items. Eye-tracking research shows that users tend to look and click at higher-ranked items, regardless of actual relevance [23]. Many click models in IR are based on document position [10]. Recently, Inverse Propensity Weighting has been explored for training unbiased relevance models in the search setting [21, 33].

Implicit feedback bias, as it pertains to recommender systems has not been deeply explored, although implicit feedback is heavily used in practice, often in an online fashion [31, 44]. There are some empirical case studies on position bias in recommender systems [15]. Several works focus on unbiased offline evaluation [19, 39] without studying how to learn unbiased models. Our work fills this gap for recommender systems and demonstrates how the training methodology can be changed. We also empirically show that weighted offline metrics (taking bias into account) align better with online metrics than traditional metrics in three real-world user-facing recommender systems.

Though less explored, addressing biased feedback in recommender system is arguably a more complicated problem than that in IR. First, real-world recommender systems typically have more diverse UI layouts than *ad hoc* search applications. As such, previous research in IR that assumes sequential browsing behavior

is typically not applicable to use cases that utilize other types of UI layouts [10]. Also, unlike IR applications where users only get access to a ranked list, in real-world applications with recommendation modules (e.g. e-commerce or video websites), users can provide implicit feedback to items beyond the recommendation UI itself, for example by simply viewing or otherwise interacting with the items outside of the recommendation UI. To the best of our knowledge, our work is the first to unify propensity estimation across different UI layouts and beyond recommendation UI-only feedback.

We also note that the Missing Not At Random (MNAR) problem has been studied for traditional recommendation problems (e.g., [30, 36]). Counterfactual inference techniques such as Inverse Propensity Weighting were also used. The key difference between MNAR and our work lies in how propensity is modelled. In our work, propensity is modelled based on *attributes* of the feedback source, ranging from the position of the item in the UI, to the user's platform (web vs. mobile), to whether the feedback was from the recommendation UI or elsewhere. In contrast, propensity in MNAR is simply modelled as predicting whether the rating is missing based on generic features using algorithms like Naive Bayes [30].

3 UNBIASED RELEVANCE MODEL LEARNING

We provide a brief review of unbiased model learning using Inverse Propensity Weighting (IPW). For more details, please refer to [24, 34]. In the following, we use a Bernoulli random variable O to denote whether the relevance of an item is observed. Without loss of generality, for item i, we use a vague notation $P(O_i = 1)$ to denote the propensity and $o_i \in \{0, 1\}$ to denote a specific value. We use $r_i \in \{0, 1\}$ to denote whether an item is relevant/attractive to a user and $[n]$ to denote the ranks of n items.

Following [24, 34], unbiased learning starts with a performance metric such as Discounted Cumulative Gain (DCG) and then derives an unbiased version of it. In this paper, we use the following variant of DCG as an example performance metric.

$$DCG = \sum_{i \in [n]} \frac{r_i}{i}$$

where r_i is binary as explained above, i is the position, and $1/i$ is the position-based discount function. With biased implicit feedback, r_i is not fully observed, so we use the IPW weights to correct the bias:

$$\overline{DCG} = \sum_{i \in [n]: o_i = 1} \frac{r_i / i}{P(O_i = 1)} = \sum_{i \in [n]: o_i = 1, r_i = 1} \frac{1/i}{P(O_i = 1)}. \quad (1)$$

The $\overline{DCG}$ metric is proven unbiased, i.e., $\mathbb{E}\{\overline{DCG}\} = DCG$ [24], and the resulting item-level IPW is

$$w_i = \frac{1}{P(O_i = 1)}.$$

Intuitively, this "weight" w_i can be assigned to an item to account for the observation bias during feedback collection. Optimizing the weighted metric on biased data results in an unbiased estimation of the metric. Previous research has shown that such a weighted metric can readily be optimized (e.g., by the LambdaLoss framework [35]).

In the unbiased learning-to-rank framework, the critical part is to estimate the propensity $P(O_i = 1)$. We next describe our framework to estimate propensity scores using offline data.

4 ATTRIBUTE-BASED PROPENSITY ESTIMATION

In this section, we show how to estimate the propensity from the implicit feedback data using an attribute-based bias model. We first set up the model and then present efficient offline methods for parameter estimation.

4.1 The bias model

We assume the observed implicit feedback Bernoulli variable C (where $C = 1$ represents a click) depends on two hidden Bernoulli variables E and R. $E = 1$ represents the event that a user examines an item, and $R = 1$ represents that the item is relevant (e.g. of interest) to a user:

$$P(C = 1|i, u, [a]) = P(E = 1|[a])P(R = 1|i, u), \quad (2)$$

where i and u are the item and user (or their feature representation), and $[a]$ is a set of attributes that affects user examination. For example, $[a]$ could represent the position of the item in the UI. In that case, this model simplifies to the position bias click model that is widely used in information retrieval [10, 29]. We will give additional context and provide concrete instantiations of these attributes $[a]$ in our use cases in Sec. 5.

Our model assumes that the examination probability only depends on the bias attributes (e.g. position) and the relevance probability only depends on the user and item. This is a common assumption [14] that we have found to work well for a number of different real-world applications. In the remainder of the paper, we utilize the following shorthand for the propensity and relevance models, respectively:

$$\theta_{[a]} = P(E = 1|[a]), \gamma_{i,u} = P(R = 1|i, u)$$

4.2 Parameter estimation

We now show how to estimate the propensity model given logged implicit feedback data. The log likelihood of regular implicit feedback data $L = (c, i, u, [a])$ is

$$\log P(L) = \sum_{c,i,u,[a] \in L} c \log \theta_{[a]}\gamma_{i,u} + (1 - c) \log(1 - \theta_{[a]}\gamma_{i,u}). \quad (3)$$

The goal is to find the best parameters $\theta_{[a]}$ and $\gamma_{i,u}$ to maximize the log likelihood. We can use EM to estimate the parameters. The EM algorithm starts from some random guess of these parameters and iteratively performs an Expectation step and an Maximization step until convergence.

Expectation step. At iteration $t + 1$ we have

$$\begin{aligned}
P(E = 1, R = 1|C = 1, i, u, [a]) &= 1 \\
P(E = 1, R = 0|C = 0, i, u, [a]) &= \frac{\theta^t_{[a]}(1 - \gamma^t_{i,u})}{1 - \theta^t_{[a]}\gamma^t_{i,u}} \\
P(E = 0, R = 1|C = 0, i, u, [a]) &= \frac{(1 - \theta^t_{[a]})\gamma^t_{i,u}}{1 - \theta^t_{[a]}\gamma^t_{i,u}} \\
P(E = 0, R = 0|C = 0, i, u, [a]) &= \frac{(1 - \theta^t_{[a]})(1 - \gamma^t_{i,u})}{1 - \theta^t_{[a]}\gamma^t_{i,u}}.
\end{aligned} \quad (4)$$

Note that other combinations are of probability 0 (e.g., $P(E = 0, R = 0|C = 1) = 0$), so we omit them here. Each quantity can be easily

calculated by using results from the Maximization step, which are randomly generated for the first iteration, as typically done for an EM algorithm.

Now we can compute marginals $P(E = 1|c, i, u, [a])$ and $P(R = 1|c, i, u, [a])$ using statistics in Eq. 4 and use these marginals in the Maximization step.

Maximization Step. At iteration $t + 1$ we have

$$\begin{aligned}\theta_{[a]}^{t+1} &= \frac{\sum_{c,i,u,[a]'} \mathbb{I}_{[a]'=[a]} \cdot (c + (1-c)P(E = 1|c, i, u, [a]))}{\sum_{c,i,u,[a]'} \mathbb{I}_{[a]'=[a]}} \\ \gamma_{i,u}^{t+1} &= \frac{\sum_{c,i',u',[a]} \mathbb{I}_{i'=i,u'=u} \cdot (c + (1-c)P(R = 1|c, i, u, [a]))}{\sum_{c,i',u',[a]} \mathbb{I}_{i'=i,u'=u}}\end{aligned} \tag{5}$$

After we run the EM process until it converges or after a fixed number of steps, the final $\theta_{[a]}$ are the propensity scores we will use for inverse propensity weighting for unbiased learning.

4.3 Regressing $\gamma_{i,u}$

Looking at Eq. 5, we note that $\theta_{[a]}^{t+1}$ is easy to calculate, since we only need to calculate one number for each attribute configuration independently and everything on the right hand side is available. For example, we only need one scalar θ for each position if we are doing position-only bias estimation (See Sec. 5.1).

However, $\gamma_{i,u}^{t+1}$ requires the exact identifiers for specific item (i) and user (u) pairs. This is challenging to estimate in practice because 1) data for each specific item-user pair is too sparse to work in the EM procedure, since the iterative EM process depends on stable estimation, and 2) the exact identifiers may not be available in the logs due to privacy (e.g. we might have the feature vector for a user-item pair in the activity log, but not their actual ids).

A natural way to address the problem is using a learning-based regression model for γ. We assume that there is a feature vector $x_{i,u}$ for each observed item i and user u pairs. We can use the same feature vector (e.g. latent factors from matrix factorization) that is used to train the actual relevance model for the application. Given $P(R = 1|c, i, u, [a])$ from the E-step, we can sample a binary label from it and use this as the label for $x_{i,u}$. This allows us to convert γ to a binary classification problem, and to use a model f_x for γ to generate calibrated probabilities that can be used in the E-step. For all of our applications, we use a depth-3 Gradient Boosted Decision Tree (GBDT) with logistic loss to implement our regression model due to its effectiveness and simplicity. Similar to [34], for iteration $t + 1$, the tree model from the last iteration is used as initialization for next iteration as the warm-start.

Remark. The goal of f_x is to estimate calibrated probabilities that can be plugged into the EM procedure for propensity estimation. Though f_x can be treated as a relevance model, the actual unbiased relevance model in our applications are learned in a second stage to optimize the ranking metric in Eq. 1 instead of log likelihood in Eq. 3. Such an approach was shown to be more effective [34]. Also, we can use more complicated model formats (e.g. Neural Network) in the second stage that can leverage larger data sets and allow careful tuning.

5 USE CASES

In this section, we describe some use cases of the proposed propensity estimation model, highlighting its practical applicability to various real-world scenarios.

5.1 Position bias estimation

Position bias is one of the most common bias factors. Our framework includes position bias estimation as a special case by setting $[a]$ to k, where k is the position of the item in the recommendation system UI. In thise case, Eq. 2 becomes

$$P(C = 1|i, u, k) = P(E = 1|k)P(R = 1|i, u), \tag{6}$$

which is the position bias click model commonly used for search use cases [14]. From this, the E-step can be derived in a straightforward manner:

$$\begin{aligned}P(E = 1, R = 1|C = 1, i, u, k) &= 1 \\ P(E = 1, R = 0|C = 0, i, u, k) &= \frac{\theta_k^t(1 - \gamma_{i,u}^t)}{1 - \theta_k^t\gamma_{i,u}^t} \\ P(E = 0, R = 1|C = 0, i, u, k) &= \frac{(1 - \theta_k^t)\gamma_{i,u}^t}{1 - \theta_k^t\gamma_{i,u}^t} \\ P(E = 0, R = 0|C = 0, i, u, k) &= \frac{(1 - \theta_k^t)(1 - \gamma_{i,u}^t)}{1 - \theta_k^t\gamma_{i,u}^t}.\end{aligned} \tag{7}$$

We omit the Maximization step since Eq. 5 can be derived similarly.

5.2 Platform-aware estimation

In real-world systems, the same recommender algorithm is often applied across different platforms, e.g. web browsers and mobile apps. The UIs and user interaction patterns often vary across platforms, often due to the various constraints imposed by the platform itself (e.g., limited screen real estate on mobile devices).

Data from various platforms is often combined and used for training a single global model. The reason for this is to reduce the complexity of having to train multiple models across platforms. However, it is possible that different platforms exhibit different bias factors. Thus it is beneficial to estimate the bias factors in a platform-aware manner.

One option is to only use randomized data in a single platform to estimate the propensity and then train the relevance model with IPW weights. Besides dropping valuable feedback data from other platforms, this may not lead to an optimal relevance model. For example, during model training, a "platform" feature might be used to account for *relevance* variance across different platforms [9]. Training a relevance model with data from a single platform will not be able to account for this factor. The second option is to run separate online randomization experiments on different platforms. However, this adds complexity and hurts the user experience even more. The third option is to run existing offline propensity estimation methods separately for each platform, which may lead to unstable estimation due to data sparseness per platform. Consistent propensity estimation across platforms allows all data to be used together with platform-specific features when estimating $\gamma_{i,u}$.

It is easy to incorporate platform-aware estimation into our framework by setting the attributes $[a]$ to be a tuple $[k, p]$, where k is the position as above, and p is the platform. Eq. 2 now becomes

$$P(C = 1|i, u, [k, p]) = P(E = 1|[k, p])P(R = 1|i, u). \tag{8}$$

We omit derivations of the EM steps due to space constraints. This formulation is quite flexible. Different platforms can have a different number of available positions or even a completely different layout, but their propensities can be estimated in a unified way.

5.3 Propensity extrapolation

Traditional bias estimation only focuses on feedback from the recommendation UI itself. One reason might be that bias estimation is more extensively studied for search scenarios, where it is query-centered and there is no clear relation between the "external feedback" and queries. However, in real-world recommender systems, there are no explicit queries and user actions beyond the recommendation UI may provide valuable feedback. For example, users of e-commerce shopping websites can provide implicit feedback by clicking and purchasing items through search or by simply browsing the website. Similarly in cloud storage systems, users can navigate independently of any document recommendations and click documents in their own folders. How can we use such feedback to train a better recommender model? Since the feedback is beyond the recommendation UI, we call this problem *propensity extrapolation*. This external feedback can be especially helpful since production recommender systems may get stuck at local optima by only using recommendation UI data.

A simple way to address this problem using our framework is to add a special pseudo *value* to the position attribute values, say k_e, for external feedback. If the item is not shown in the UI, its position attribute will have the value k_e. For example, in the M-step, we will calculate propensity estimates for this special position as

$$\theta_{k_e}^{t+1} = \frac{\sum_{c,i,u,k'} \mathbb{I}_{k'=k_e} \cdot (c + (1-c)P(E=1|c,i,u,k_e))}{\sum_{c,i,u,k'} \mathbb{I}_{k'=k_e}} \quad (9)$$

The rest of our EM algorithm is the same after adding this new attribute value. This demonstrates the flexiblity of our attribute-based propensity model.

Implementation. We now provide guidelines on how to utilize this external feedback in practice. The key part is proper logging of both the model features and feedback. For simplicity, let us assume that we have a single UI and this UI has k positions.

A common recommendation process is to first generate $N > k$ candidates, score them using the relevance model, and display the top k results in the UI [31]. The candidate generation process varies across applications. For example, in an e-commerce website category (e.g. home decorations) page, the candidates could be all the items with available inventory in the category, or the most popular items in the last few days. A complete treatment on the candidate generation process is beyond the scope of the paper.

There are two key steps: 1) log the features for all or a uniform sample of the N candidates; 2) log the user's behavior post recommendation. If positive feedback is provided for an item in the N candidates in the same session (e.g. within 5 minutes), the positive label should be joined with the item. Note that this applies to feedback both inside and outside the recommendation UI.

Since we only log features for the N items, not all of the external feedback from outside of the recommendation UI can be joined back. A metric called "coverage" can be introduced and monitored as the fraction between the number of sessions with positive feedback on the N candidates over the total number of sessions with positive feedback. The higher the coverage, the more accurate the propensity estimation will be due to the additional feedback. However, there is usually a trade-off between coverage and resource costs. For example, if the candidate set contains all items available in an application, coverage will always be 1, but the inference and logging cost could be high. Finding the right trade-off is an independent research problem that can be explored as future work.

5.4 Discussion

We showed three simple but practical use cases using our propensity estimation framework. We note that different attributes could be easily combined. For example, in Sec. 5.2 we showed platform and position could be combined into a tuple. In one of our applications shown later, we combine position, platform, and feedback extrapolation. More attributes can also be incorporated, such as feedback source (with values "from search", "from browsing", etc.), as long as they might contribute to presentation bias and there is reasonable amount of data (which is one advantage of using implicit feedback). Thus, our framework allows to study more complex attributes and we leave it as future work.

6 CASE STUDIES

We demonstrate how our framework helps to improve three real-world recommendation systems, each involving millions of users.

6.1 Overview

We aim to validate the generality and effectiveness of our framework. The three applications presented here have different UI layouts, item types, and underlying relevance model learning algorithms. An overview can be found in Tbl. 1.

Table 1: Overview of the three case studies.

Name	Item Type	Layout	Model
Zero State Search	query	vertical	GBDT
Shared With Me	people&document	grid	GBDT
Quick Access	document	horizontal	neural network

For each application, we first estimate and examine propensity scores using our method on logged data. Then we train an experimental model with IPW, where the only difference from the production baseline is the usage of propensity scores. Besides using data from the same date ranges, we also use the same hyperparameters (such as the number of layers in the neural network) as the baseline model. These hyperparameters have been well-tuned for the baseline model, which was deployed in production and had undergone multiple quality iterations, so we may get even better results if we do more careful tuning with the experimental models. Since our focus is to improve the model without changing the features or model architecture, we omit some details of feature engineering due to space constraints.

After offline evaluation, we conduct standard online A/B experiments for each application. Each experiment involves millions of users and runs for approximately 2 weeks. The models have been deployed in production systems for an extended amount of time due to the positive metrics and ease of deployment, since our framework estimates propensity offline and IPW trained models do not require any additional serving infrastructure support.

6.2 Evaluation Metrics

6.2.1 Offline metrics. We use both (traditional) unweighted Mean Reciprocal Rank (MRR) and Weighted MRR (WMRR) as our offline evaluation metrics, which are specifically defined as

$$\text{MRR} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{\text{rank}_i}, \text{WMRR} = \frac{1}{\sum_{i=1}^{N} w_i}\sum_{i=1}^{N} w_i \frac{1}{\text{rank}_i}.$$

where $rank_i$ is the minimum rank of the clicked items for the i-th list and N is the total number of lists in the evaluation data set. The WMRR metric is an unbiased version of MRR and is extensively used in related work [33]. w_i is the learned IPW weight for each positive item. WMRR gives a higher reward (assuming $w > 1$) to models that improve the rank of hard-to-find positive items during the data collection phrase.

6.2.2 Online metrics. In the following, we report a slightly different set of metrics for each application since different products track different online metrics due to having different intended use cases. One common online metric is Click-Through Rate (CTR). This metric can be tracked for online experiments, but is not applicable for offline evaluation. In fact, WMRR on biased data or MRR on unbiased data is a proxy for online metrics like CTR to measure the utility of a ranked list offline. We will show that offline WMRR is correlated with online metrics better than MRR.

6.3 Zero State Search

6.3.1 Introduction. The Google Drive Zero State Search feature is intended to save Drive search users effort by recommending queries and other refinements that are considered relevant to them, even before any query is issued by the user. See Fig. 2 for an illustration and [4] for more information. In this work we only focus on the query recommendation section, which has a vertical layout and shows up to 3 suggested queries when a user clicks the search box. The query candidates are generated synthetically from n-grams found in each user's documents.

To train the ranking model, we use GBDT with around 20 features. At the time of testing our model, the production model had gone through several quality iterations, including feature additions and hyper-parameter tuning. The GBDT model is of depth 3 and runs for 1000 steps. The features include simple features such as the length of the n-gram, and more complicated features such as the number of times a user interacted with documents containing that n-gram in the last 7 days. Both the production model and our experiment model use 3 weeks of offline data in the same date range, consisting of millions of clicked sessions.

6.3.2 Learned propensity. We use the first week of training data for propensity estimation. For this use case we use the position bias estimation described in Sec. 5.1. Since we have a vertical layout, it is straightforward to just use 3 values for the positions. The learned inverse propensity weights are shown in Fig. 3. The weights are intuitive, as higher weights will be assigned to items that are further away from the search box.

6.3.3 Offline evaluation. We use both MRR and WMRR for offline evaluation. The comparative results on 1 week of test data are shown in Tbl. 2. We can see the IPW model is neutral in MRR but positive in WMRR when compared with the production baseline.

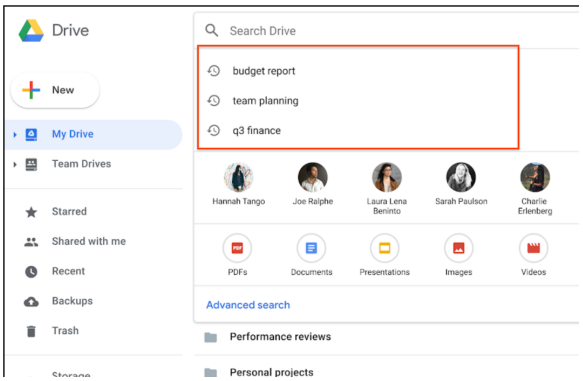

Figure 2: An illustration of Google Drive's Zero State Search from [4], which recommends queries, collaborators, and others. In this work we only focus on the query section.

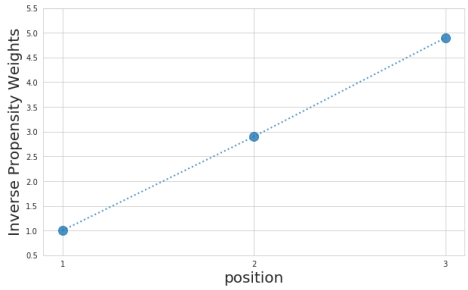

Figure 3: Learned propensity weights for Zero State Search.

Table 2: Offline relative performance compared with the (unweighted) production baseline. † denotes a statistically significant difference at the $p < 0.05$ level using a two-tailed t-test.

Model	MRR	WMRR
IPW model	+0.05%	+0.68%†

6.3.4 Online evaluation. Real-world online experimental results are shown in Tbl. 3. We report several key metrics including CTR and Search User Engagement (percentage of Drive users who click on a Zero State Search suggestion). We can see that the IPW model can achieve significantly better metrics than the baseline. The improvements are quite meaningful in our production setting. We note that online metrics align better with WMRR than MRR here. We will return to discuss this more discussion in Sec. 6.6.

Table 3: Online relative performance compared with the production baseline for Zero State Search. † denotes a statistically significant difference.

Model	CTR	Engagement
IPW model	+2.83%†	+1.17%†

6.4 Shared With Me

6.4.1 Introduction. The "Shared with Me" section recommendation module in Google Drive recommends relevant (shared) document-people pairs to a user, so that the user can click on a document to quickly open it. Instead of a list, the UI has a grid layout and shows up to 8 document-people pairs. See Fig. 4 for an illustration and [3] for more information.

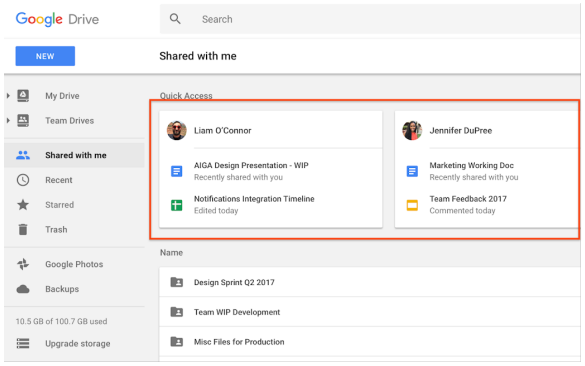

Figure 4: An illustration of Google Drive's Shared With Me from [3], which recommends document-people pairs. Note that although "Quick Access" is shown on-top, it is a different product than the Quick Access covered in Sec. 6.5.

To train the ranking model, we use GBDT with around 20 features for each document-people pair. The GBDT model is of depth 5 and runs for 800 steps. The features focus on document sharing behaviors such as when the document was shared and how many people it was shared with. It also uses person and document importance scores that are generated by other internal systems. Both the production model and our experiment model are trained over 3 weeks of data in the same data range, consisting of millions of clicked sessions.

6.4.2 Learned propensity. We use the first week of training data for propensity estimation. For this use case we use the position bias estimation described in Sec. 5.1. To tackle the grid layout, we simply use 8 categorical position values. The learned inverse propensity weights are shown in Fig. 5. We can see a symmetric layout along the diagonal axis.

	Position 1	Position 2	Position 3	Position 4
Document 1	1	1.61	2.45	6.45
Document 2	1.62	2.51	3.75	8.3

Figure 5: The learned propensity weights for Shared With Me with heatmap.

6.4.3 Offline evaluation. We also use MRR and WMRR as the offline evaluation metrics on 1 week of test data. The comparative results are shown in Tbl. 4. We note that the IPW model is worse than the baseline on unweighted metric MRR, but better on the weighted metric.

Table 4: Offline relative performance compare with the (unweighted) production baseline. † denotes a statistically significant difference.

Model	MRR	WMRR
IPW model	-1.24%†	+1.19%†

6.4.4 Online evaluation. We report real-world online experimental results in Tbl. 5. We can see that the experimental model can achieve significantly better CTR than the baseline. The improvements are quite significant in the production setting. We note that online metrics once again align better with WMRR than MRR. This further shows the importance of using unbiased IPW-based metrics for offline evaluations.

Table 5: Online relative performance compared with the production baseline for Shared With Me. † denotes a statistically significant difference.

Model	CTR
IPW model	+1.92%†

6.5 Quick Access

6.5.1 Introduction. Quick Access [1, 2, 12, 32] is a feature available in both the mobile and web versions of Google Drive that provides users a shortcut to their most relevant files. Improving Quick Access recommendations is important because of the large amount of traffic it receives on a daily basis and how much time it saves users. As [32] has demonstrated, using Quick Access gets users to their files around 50% faster than if they had not. See Fig. 1 for an illustration of the web and mobile views of Quick Access.

There are several hundred features for the Quick Access model. Example features include users' past behavior, such as when and how many times a user interacted (create/edit/comment) with a document. Other features include collaborative ones (e.g., how frequently other users interacted with a shared document) and contextual ones (e.g., platform, time of day, etc.). We use a deep neural network implemented in Tensorflow [5] for the relevance model. At the time of our experiment, the model had been well-tuned with several recent techniques deployed, including latent cross [9, 28], residual block [17], and batch normalization [22]. An illustration of the model architecture is shown in Fig. 6.

Also note that the production model already uses external feedback gathered beyond the UI, but with a unit weight. This was shown to be highly beneficial, resulting in a 2% CTR increase. The gain was not due to more data since data was sampled for a fair comparison. The current work demonstrates how assigning a learned propensity weight to the external feedback can further improve performance.

6.5.2 Learned propensity. We use the first week of training data for propensity estimation. For this use case we combine position bias, platform-aware, and propensity extrapolation attributes introduced in Sec. 5.1, Sec. 5.2, and Sec. 5.3, respectively. We use both web and mobile data because it was previously found that using "platform" as a feature as a latent cross feature could significantly improve

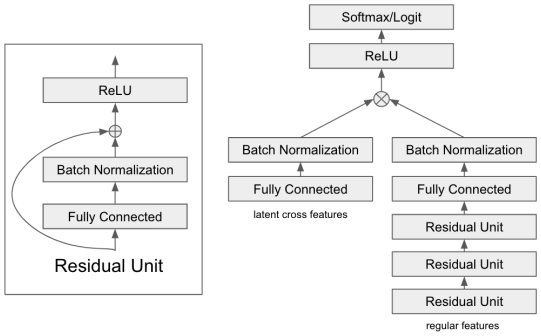

Figure 6: Quick Access model architecture. Regular features and latent cross features are fed into different towers can merged by latent cross [9]. The basic building blocks are residual units [17].

performance. Using data from one platform will make that feature useless. Fortunately, platform-aware propensity estimation in our framework allows for unified propensity estimation across all data. For propensity extrapolation, we set the size of candidates as $N = 100$ in the online serving system.

At the time of testing, Quick Access showed up to 5 documents in each platform's UI. We assign 6 position attribute values (5 position values plus the special out-of-UI value k_e) for each platform. The learned inverse propensity weights are shown in Fig. 7. We can see that the learned weights are intuitive, e.g., mobile users need more effort to find non-UI documents possibly due to the smaller screen.

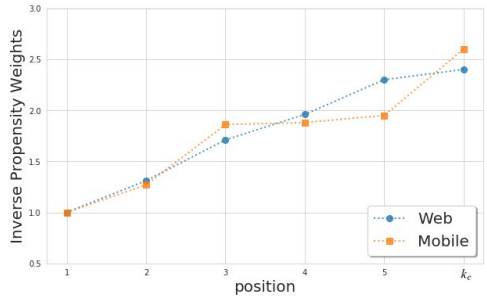

Figure 7: The learned propensity weights for Quick Access on both web and mobile data.

6.5.3 Offline evaluation. We show the comparative offline metrics on heldout data during model training in Fig. 8. It is clear that, similar to previous applications, the production baseline model performs better on traditional unweighted metrics, but is inferior on weighted metrics.

6.5.4 Online evaluation. We report real-world online experimental results in Tbl. 6 for both web and mobile traffic. We can see that the experimental model can get significantly better CTR than the baseline on both platforms. The improvements are quite significant

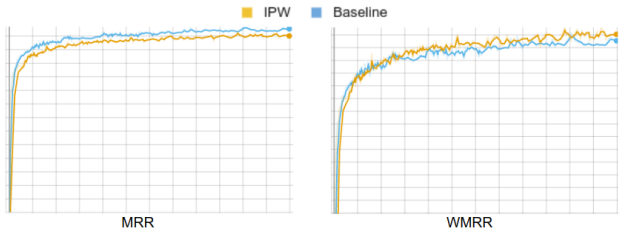

Figure 8: Comparative offline MRR and WMRR metrics on heldout data during model training of IPW and baseline models for Quick Access. The WMRR curves are more shaky than MRR curves since validation is performed on randomly selected data batches that could possess different weights.

in the production setting and we also note that online metrics align better with WMRR than MRR.

Since Quick Access is powered by a deep neural network, we also ran experiments using the approach proposed by [45], where the selection bias is reduced by adding a shallow tower to the main model that estimates the propensity score. The position is fed in during the training process (with a 10% feature drop-out rate to prevent the model from over-dependence on the position feature). This proposed method also can estimate propensity scores in an offline manner and served as a fair comparison to our approach. We experimented with a variety of different drop-out rates ranging from 0% drop-out to 50% drop-out. We found that all experiments with this method resulted in no statistical significant positive gain.

Table 6: Online relative performance compared with the production baseline for Quick Access. † denotes a statistically significant difference.

Model	Web CTR	Mobile CTR
IPW model	+0.94%†	+1.42%†

6.6 Remark on offline evaluation metrics

We remark on the importance of unbiased offline evaluation metrics, which was also covered by other recent research [19, 39]. Using accurate offline metrics that align with online metrics is important for both model tuning and deciding which candidate models should make it to the online experimentation phase. From all 3 applications, we observe that weighted metrics using the estimated inverse propensity weights align better with real-world online metrics. It is also clear that traditional unweighted metrics can be misleading sometimes. Note that due to our particular way of optimizing weighted metrics using IPW, we tend to get inferior unweighted metrics offline. This does not mean unweighted metrics are always misleading. For example, it is possible to improve both unweighted and weighted offline metrics by adding a new powerful feature.

We have shown empirically that our framework provides a natural way of optimizing weighted metrics with weighted training, since we focus on learning propensity weights that can be used for both objectives. However, we acknowledge that there could be other unbiased metrics and different ways to optimize them offline,

and believe that unbiased model training and unbiased evaluation can be treated as separate research topics.

7 CONCLUSION

In this work, we proposed a novel attribute-based propensity estimation framework for unbiased learning in recommender systems. Besides generalizing traditional position bias models, our framework allows for unifying propensity estimation across different platforms, going beyond simple UI-only feedback, and using an offline process that does not hurt the user experience nor requires significant serving system changes. On three real-world recommender systems, we show that IPW training with attribute-based propensity estimation can significantly outperform well-tuned production baselines using large-scale real-world A/B experiments that involve millions of users. We also note the importance of optimizing unbiased metrics when training the relevance models.

Our work opens up a few potential directions for future work, including: (1) how to identify informative attributes for different applications is an interesting research question; (2) in recommender systems, non-UI feedback is important and how to derive finer attributes is worth exploring more; and (3) it is also interesting to see how to apply our methods to estimate propensity for traditional recommendation problems with explicit feedback that is in general not missing at random.

REFERENCES

[1] 2016. Save time with Quick Access in Drive. https://gsuiteupdates.googleblog.com/2016/09/save-time-with-quick-access-in-drive.html. Accessed: 2020-01-15.

[2] 2017. Quick Access in Google Drive now available on the web. https://gsuiteupdates.googleblog.com/2017/05/quick-access-in-google-drive-now.html. Accessed: 2020-01-15.

[3] 2018. Find shared content with new file organization in Google Drive. https://gsuiteupdates.googleblog.com/2018/03/find-shared-content-with-new-file.html. Accessed: 2020-1-15.

[4] 2018. New intelligent search box in Google Drive. https://gsuiteupdates.googleblog.com/2018/07/intelligent-search-box-in-google-drive.html. Accessed: 2020-01-15.

[5] Martin Abadi et al. 2016. TensorFlow: A system for large-scale machine learning. In *OSDI*. 265–283.

[6] Aman Agarwal, Ivan Zaitsev, Xuanhui Wang, Cheng Li, Marc Najork, and Thorsten Joachims. 2019. Estimating Position Bias without Intrusive Interventions. In *WSDM*. 474–482.

[7] Qingyao Ai, Keping Bi, Cheng Luo, Jiafeng Guo, and W. Bruce Croft. 2018. Unbiased Learning to Rank with Unbiased Propensity Estimation. In *SIGIR*. 385–394.

[8] James Bennett, Stan Lanning, et al. 2007. The Netflix prize. In *Proceedings of KDD cup and workshop*.

[9] 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 *WSDM*. 46–54.

[10] Alexey Borisov, Ilya Markov, Maarten de Rijke, and Pavel Serdyukov. 2016. A Neural Click Model for Web Search. In *WWW*. 531–541.

[11] Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed H. Chi. 2019. Top-K Off-Policy Correction for a REINFORCE Recommender System. In *WSDM*. 456–464.

[12] Suming J. Chen, Zhen Qin, Zac Wilson, Brian Calaci, Michael Rose, Ryan Evans, Sean Abraham, Donald Metzler, Sandeep Tata, and Mike Colagrosso. 2020. Improving Recommendation Quality in Google Drive. In *KDD*.

[13] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Workshop on Deep learning for Recommender Systems*. 7–10.

[14] Aleksandr Chuklin, Ilya Markov, and Maarten de Rijke. 2015. *Click Models for Web Search*. Morgan & Claypool.

[15] Andrew Collins, Dominika Tkaczyk, Akiko Aizawa, and Joeran Beel. 2018. Position bias in recommender systems for digital libraries. In *International Conference on Information*. Springer, 335–344.

[16] Zhichong Fang, A. Agarwal, and T. Joachims. 2019. Intervention Harvesting for Context-Dependent Examination-Bias Estimation. In *SIGIR*.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*. 770–778.

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

[19] Katja Hofmann, Anne Schuth, Alejandro Bellogin, and Maarten De Rijke. 2014. Effects of position bias on click-based recommender evaluation. In *ECIR*. 624–630.

[20] Y. Hu, Y. Koren, and C. Volinsky. 2008. Collaborative Filtering for Implicit Feedback Datasets. In *ICDM*. 263–272.

[21] Ziniu Hu, Yang Wang, Qu Peng, and Hang Li. 2019. Unbiased LambdaMART: An Unbiased Pairwise Learning-to-Rank Algorithm. In *WWW*. 2830–2836.

[22] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[23] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, Filip Radlinski, and Geri Gay. 2007. Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search. *ACM Trans. Inf. Syst. (TOIS)* (2007), 1–7.

[24] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. 2017. Unbiased Learning-to-Rank with Biased Feedback. In *WSDM*. 781–789.

[25] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 8 (2009), 30–37.

[26] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. 2010. A Contextual-bandit Approach to Personalized News Article Recommendation. In *WWW*. 661–670.

[27] 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 *WWW*. 583–592.

[28] Zhen Qin, Zhongliang Li, Michael Bendersky, and Donald Metzler. 2020. Matching Cross Network for Learning to Rank in Personal Search. In *WWW*. 2835–2841.

[29] Matthew Richardson, Ewa Dominowska, and Robert Ragno. 2007. Predicting Clicks: Estimating the Click-through Rate for New Ads. In *WWW*. 521–530.

[30] Tobias Schnabel, Adith Swaminathan, Ashudeep Singh, Navin Chandak, and Thorsten Joachims. 2016. Recommendations as Treatments: Debiasing Learning and Evaluation. In *ICML*. 1670–1679.

[31] Jiaxi Tang and Ke Wang. 2018. Personalized top-n sequential recommendation via convolutional sequence embedding. In *WSDM*. 565–573.

[32] Sandeep Tata et al. 2017. Quick Access: Building a Smart Experience for Google Drive. In *KDD*. 1643–1651.

[33] Xuanhui Wang, Michael Bendersky, Donald Metzler, and Marc Najork. 2016. Learning to Rank with Selection Bias in Personal Search. In *SIGIR*. 115–124.

[34] Xuanhui Wang, Nadav Golbandi, Michael Bendersky, Donald Metzler, and Marc Najork. 2018. Position Bias Estimation for Unbiased Learning to Rank in Personal Search. In *WSDM*. 610–618.

[35] Xuanhui Wang, Cheng Li, Nadav Golbandi, Mike Bendersky, and Marc Najork. 2018. The LambdaLoss Framework for Ranking Metric Optimization. In *CIKM*. 1313–1322.

[36] Xiaojie Wang, Rui Zhang, Yu Sun, and Jianzhong Qi. 2019. Doubly Robust Joint Learning for Recommendation on Data Missing Not at Random. In *ICML*. 6638–6647.

[37] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. 2017. Recurrent recommender networks. In *WSDM*. 495–503.

[38] Qingyun Wu, Hongning Wang, Liangjie Hong, and Yue Shi. 2017. Returning is believing: Optimizing long-term user engagement in recommender systems. In *CIKM*. 1927–1936.

[39] Longqi Yang, Yin Cui, Yuan Xuan, Chenyang Wang, Serge Belongie, and Deborah Estrin. 2018. Unbiased Offline Recommender Evaluation for Missing-not-at-random Implicit Feedback. In *RecSys*. 279–287.

[40] Xing Yi, Liangjie Hong, Erheng Zhong, Nanthan Nan Liu, and Suju Rajan. 2014. Beyond Clicks: Dwell Time for Personalization. In *RecSys*. 113–120.

[41] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *KDD*. 974–983.

[42] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)* (2019), 5:1–5:38.

[43] Qian Zhao, Shuo Chang, F Maxwell Harper, and Joseph A Konstan. 2016. Gaze prediction for recommender systems. In *RecSys*. 131–138.

[44] Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2018. Deep reinforcement learning for page-wise recommendations. In *RecSys*. 95–103.

[45] 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 *RecSys*. 43–51.