Downside Management in Recommender Systems

Huan Gui† Haishan Liu‡ Xiangrui Meng† Anmol Bhasin ‡ Jiawei Han†
† University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA
‡ LinkedIn Corp, Mountain View, CA 94043 USA
†{huangui2, hanj}@illinois.edu ‡{haliu, ximeng, abhasin}@linkedin.com

Abstract—In recommender systems, bad recommendations can lead to a net utility loss for both users and content providers. The downside (individual loss) management is a crucial and important problem, but has long been ignored. We propose a method to identify bad recommendations by modeling the users’ latent preferences that are yet to be captured in a residual model, which can be learned independently on top of existing recommendation algorithms. We include two components in the residual utility: benefit and cost, which can be learned simultaneously from users’ observed interactions with the recommender system. We further classify user behavior into fine-grained categories, based on which an efficient optimization algorithm to estimate the benefit and cost using Bayesian partial order is proposed. By accurately calculating the utility users obtained from recommendations based on the benefit-cost analysis, we can infer the optimal threshold to determine the downside portion of the recommender system. We validate the proposed method by experimenting with real-world datasets and demonstrate that it can help to prevent bad recommendations from showing.

I. INTRODUCTION

Recommender systems seek to find the right items to recommend to users. The task can be viewed as a way to predict a personalized ranking on a set of items [16]. A myriad of applications of recommender systems exist in domains such as e-commerce [17], movie streaming [2], social network [20], and so on. Accurate recommendations give users the ability to navigate more efficiently. As a specific example, a job posting recommender system may recommend a “senior software engineer” among millions of positions in stock to a user who works in the software industry. Such recommendations can be made by finding user-item pairs that meet some criteria of relevance or follow certain business rules.

Recommender systems are typically evaluated by and tuned to optimize metrics that characterize the upside (overall gain) of the system, such as the click-through-rate (CTR) or the conversion rate, while little attention is given to the potential downside risk (individual loss). A system may appear to perform well in general, maintaining CTR at a satisfactory level, but contain a portion of inferior results that might repel certain users. Moreover, it is often the case that bad results perceived by users can be perfectly reasonable ones to the system according to the objective or the specific relevance model it is based upon. For example, recommending an entry-level job position to someone currently with a senior-level position would not make a good user experience and can be offensive, but it may appear perfectly normal to a system that does not consider job seniority. Another more subtle example is that it is highly subjective to determine the merits of proposing a more senior position at a lesser company. One way to mitigate the problem is to augment the recommendation model to encompass additional features that may avoid the observed failure cases. However, this process is ad-hoc, and sometimes infeasible, depending on the current model used, and it may be ineffective in some scenarios. For instance, for a user with a senior-level position in a very specialized field, the highest ranked job recommendations might all be entry-level ones simply because these are the only available ones at the time. Another scenario is to recommend items to users who have absolutely no need to consume such items, such as recommending jobs to corporate CEOs. When users have no interest, any recommendations may be distracting or even annoying. In both scenarios, the best recommendations are actually bad ones. Bad recommendations may cause user experience deterioration and eventually customer attrition, as they provide no help to users on information filtering, but rather waste users’ attention, an extremely important resources for online monetization [9]. Therefore, we find the downside management of recommender systems to be a very important yet challenging problem. The challenges are summarized as follows:

- It is hard to identify and model bad recommendations.
- Downside management should be independent of underlying recommendation algorithms.
- Personalization is an important issue in downside management.

Existing recommendation systems are built upon a variety of models, while the downside management is a common problem. The algorithm developed for downside management should be easily applied to existing recommendation models.

To solve this problem, we propose a residual model based on the benefit-cost formulation to capture users’ utility missed in the original recommendation model. Parameters in the residual
model are estimated based on analyzing users’ historical behavior using an efficient algorithm of alternating maximization with accelerated gradient descent. A personalized threshold is learned for each user. If the utility of a recommendation falls below the threshold, then it is considered a net loss towards the viewing user’s experience, and the system can opt to simply not show the recommendation.

We make the following contributions in this work:
- We study the downside management problem in recommender systems, where best candidates can be bad ones.
- We propose a benefit-cost model that captures users’ residual utility. The model can be applied on the top of existing recommendation algorithms.
- We propose a unified learning framework that combines both classification and ranking and is able to infer an optimal personalized threshold for each user to determine the quality of the recommendations.
- We solve the residual model as an optimization problem that can be approached efficiently by alternating maximization with accelerated gradient descent.
- We have tested our model in large-scale real-world recommendation applications, demonstrating the power and feasibility of the proposed method.

The remainder of the paper is organized as the follows.

In Section II, we categorize users’ feedback based on Pareto improvement for behavior modeling. We present the benefit-cost recommendation algorithm in Section III, followed by the learning algorithm in Section IV. The experiments are carried out in Section V, and related work is discussed in Section VI. Finally, Section VII concludes the work.

II. USER BEHAVIOR MODELING

Generally speaking, there are two types of situations that may cause top ranked recommendations (according to the model) to become bad ones (perceived by users): 1) there is no good items available that suit users’ need, or 2) users have no interest in consuming items of certain types. In either case, even with enough slots to show items, a downside management algorithm should hide those recommendations from users and show only those that meet a certain standard. Most recommender systems produce recommendations as a list of items ranked by some scoring function. Therefore, one way to achieve the downside management is to find a threshold for the score to decide whether to show the recommendation or not.

We believe the threshold should not be part of the underlying recommendation model for two reasons. First, the perceived bad aspect of a recommendation may not be able to be explained by the model. In order to identify these bad recommendations, it may require the model’s objective, feature space, and the training data to be significantly changed. It might be infeasible when there are multiple recommender systems, in which case each recommender system needs to be specifically tuned to tackle the downside management. Secondly, even if we were able to change the premise of the model, the scalability and flexibility of the model would be undermined whenever there is a new aspect needed to be incorporated, as the structures of underlying recommender system has been altered. Therefore, in stead of modifying the underlying recommendation models, we propose a residual model that can be applied independently on top of existing recommendation algorithms. To be more specific, instead of managing the downside of recommendations from scratch and integrating it with the underlying recommender system, we model it as a post-process of underlying recommender system. By modeling downside management as a post-process module, we can not only preserve structure and workflows of the underlying recommender system, but also enjoy the flexibility when applying the same downside management models to an ecosystem of recommender algorithms.

We explicitly model users’ latent preferences by two groups of parameters, benefit and cost. The parameters would be learned from historical interaction behavior with recommended items. As the downside management is post-processing of underlying recommender system, we expect the parameters will compensate the existing scores. The benefit and cost model can also be nicely explained by the benefit-cost model which originated from microeconomics analysis. Firstly, we assume people are rational utility maximizers [18], i.e., users would rationally maximize their utility; Secondly, the utility is modeled as the extend to which users are interested to the recommended items; Thirdly, the cost is modeled as how much people value their time and attention; Finally, for each recommended items, users net utility would be defined as utility minus cost, and only recommendation with positive net utility would be recommended. The idea of utility maximization would greatly benefit and simplify the choice of global threshold as we will discuss later. Remark that a high-cost user is generally more critical, less tolerant to bad recommendations, and has a higher hurdle to overcome in order to act positively on a recommended item.

We first summarize the notations, followed by the modeling of users historical behaviors.

A. Notation

We use $u$ to denote a user, $i$ for an item, $(u,i)$ for a user-item relationship, representing one’s feedback/assessment for the item, and $R$ for the set of all user-item relationships. Define utility function $g_u(i)$ to measure the overall utility $u$ can gain by interacting with item $i$. Utility function drives users’ decision making. For example, given two options $i_1$ and $i_2$, based on users’ behavior analysis, a user $u$ would favor the one that gives better utility, i.e., $u$ would choose $i_1$ if $g_u(i_1) > g_u(i_2)$, or $i_2$ otherwise.

B. Modeling the Downside

1) Behavior Categories: Many recommender systems define users’ feedback on recommendations as either positive or negative. Positive feedback include click-through, conversion, etc., while negative feedback include deletion (if applicable), ignoring the recommendation, etc. In order to capture the
harmful downside, we propose a finer-grained classification of users’ behavior as following:

\[ \mathcal{R} = \mathcal{R}_{++} \cup \mathcal{R}_+ \cup \mathcal{R}_- \cup \mathcal{R}_{--} \]

where \( \mathcal{R}_{++}, \mathcal{R}_+, \mathcal{R}_-, \mathcal{R}_{--} \) are for strongly positive feedback, weak positive feedback, weak negative feedback, and strong negative feedback, respectively.

As it is nearly impossible to obtain the label data for downside management, we use the strongly negative feedback as surrogate for labels in downside system. We will show in experimental part that this strategy works well to capture future catastrophic recommendations, even with limited number of strongly negative feedback labels. We also call such downside recommendations catastrophic as they may cause user attrition, the worst outcome of a recommender system, in which case the terminologies of catastrophic recommendation detection and downside management would used interchangeably.

With regard to the recommendation in real-world cases, we give out some possible activities in each of these categories for better understanding. A typical example on e-commerce system is shown as follows:

1. \( \mathcal{R}_{++} \): user selects/watches/purchases \( i \) (conversion);
2. \( \mathcal{R}_+ \): user clicks on \( i \) and views the content of \( i \), but does not convert in the viewing session;
3. \( \mathcal{R}_- \): user has interactions with some items (where multiple recommendations are available), but ignores \( i \);
4. \( \mathcal{R}_{--} \): user explicitly shows negative feedback on \( i \), such as deleting or complaining about the recommendation.

The interaction between users and items can also be classified by other schemes. In our case, the strongly negative feedback can be viewed as users’ interaction with the recommended item to stop the same item from showing again, if possible. The criteria of different classification schema would be given in the remaining part of this section.

2) Pareto Improvement on the Downside: We introduce an other tool called Pareto improvement [22] to standardize the choice of different schemas for behavior categorization, which would also lead to tractability of catastrophic recommendations detection.

Pareto improvement is based on the concept of Pareto optimality, also known as Pareto efficiency, which is a state of allocation of resources that it is impossible to improve the utility of one individual without harming at least one other individual’s utility. Given the initial allocation of resources, if the utility of one individual can be improved, without making any other individual’s utility less, this kind of improvement is called Pareto improvement. With Pareto improvement, we can build inequalities between the behavior categories and a new imaged interaction category between users and items, defined as follows.

When users receive very bad recommendations, they may provide strong negative feedback so that this kind of recommendations would not show up again. That is to say, when given a choice, user would prefer not seeing the bad recommendations at all. Therefore, we define a new type of relationship between users and items: \( \mathcal{R}_0 \), indicating that the recommended item is hidden and user will not see the item. Note that this is an imaged situation as the user has already seen the items, otherwise we will not have the historical records of this specific (user, item) pair.

Now, we will take a closer look at user’s strongly negative feedback. Suppose user \( u \) has given strongly negative feedback to item \( i \). The null recommendation can be understood as instead of being recommended by item \( i \), user \( u \) is being recommended by a null recommendation \( \phi \). Based on previous analysis, as user \( u \) has taken action to prevent \( i \) from showing up, user \( u \) has explicit preference of the null recommendation \( \phi \) over the existing recommendations \( i \). This state can also be viewed as user’s interaction with a null recommendation \( \phi \), i.e., \( (u, \phi) \in \mathcal{R}_0 \). When a recommendation \( i \) is hidden from user \( u \) (in other words, a \( \phi \) recommendation is shown to \( u \)), the overall utility \( u \) obtains is \( g_u(\phi) = 0 \).

Based on the Pareto analysis, for any pair \( (u, i) \) in the catastrophic recommendation category \( \mathcal{R}_{--} \), we have \( g_u(i) < g_u(\phi) = 0 \), as user \( u \) prefers no recommendations to the shown ones. We shorthand \( \mathcal{R}_+ \subset \mathcal{R}_- \) for the case that \( g_u(i) < g_u(j), \forall i \in \mathcal{R}_+, j \in \mathcal{R}_- \). So, we have \( \mathcal{R}_- \prec \mathcal{R}_\phi \).

By applying Pareto analysis, we can get the global threshold from the rigid analysis via defining the null status. Similarly, \( \mathcal{R}_\phi \prec \mathcal{R}_{++} \), and \( \mathcal{R}_\phi \prec \mathcal{R}_+ \).

To note that the relationship cannot be applied to the category of \( \mathcal{R}_- \), as we cannot build the inequality from Pareto analysis, that \( \forall (u, i) \in \mathcal{R}_- \), the preference order cannot be inferred as user \( u \) did not take any actions to \( i \), which is equivalent to \( \phi \).

Putting pieces together, we have the following relations:

\[ \mathcal{R}_- \prec \mathcal{R}_- \prec \mathcal{R}_\phi \prec \mathcal{R}_+ \prec \mathcal{R}_{++} \]  

(1)

In the following section, we model the ranking order by the residual model.

III. Benefit-Cost Model

In this section, we introduce our benefit-cost model, which is a residual model can be built on the top of any existing recommendation models. As in a residual model, we split the overall utility \( g_u(i) \) that a user \( u \) can gain from interacting with an item \( i \) into two parts: partial utility, which is estimated by the underlying recommendation algorithm, and the residual utility, which is estimated from the benefit-cost model:

\[ g_u(i) = s_u(i) + f_u(i), \]  

(2)

where \( s_u(i) \) is the existing utility of recommending \( i \) to \( u \) estimated by the underlying recommending algorithm, and \( f_u(i) \) is the residual utility function of recommending \( i \) to \( u \) that is not captured in \( s_u(i) \). Details of both components of the utility function are described below.

A. Existing Utility

The amount of useful information captured by the existing recommender system is reflected in the scores it produces to rank items. Most current recommendation algorithms are based...
on some notion of relevance or proximity. The output of such systems, i.e., the ranking score, can be used as a proxy variable for the existing utility. The existing utility function of \( s_u(i) \) is defined as the following:

\[
s_u(i) = s_{ui} I_{i \in I} + 0 I_{i = \phi},
\]

where \( I \) is the item set, \( I \) is the indicator function, and \( s_{ui} \in \mathbb{R} \) is the ranking score returned by underlying recommendation algorithms. When the recommendation is \( \phi \), the existing utility is defined to be 0.

### B. Residual Utility

The residual utility function models users’ additional gain of utility, which can be either positive or negative, that is failed to be modeled in the underlying recommender systems. We further decompose the residual utility function into the benefit function and the cost function as follows:

\[
f_u(i) = b_u(i) - c_u(i),
\]

where \( b_u(i) \) models \( u \)'s latent preference for item \( i \) that is not captured by the underlying recommendation algorithm, and \( c_u(i) \) models the user-dependent cost one has to pay in order to make any positive feedback towards the recommendation.

1) Benefit Function: The benefit function \( b_u(i) \) characterizes the positive residual utility one can gain due to the uncaptured latent features. It is a user-dependent function with \( |U| \times |I| \) values to be estimated, where \( U \) and \( I \) are the user set and item set respectively. To reduce the number of parameters in the learning process and to avoid overfitting, we factor \( b_u(i) \) into the inner products of \( \{W_i\} \subset \mathbb{R}^d \), latent feature vectors of items, and \( \{V_u\} \in \mathbb{R}^d \), those of users, where \( d \) is the dimension of latent feature space. The degree of freedom is hence \( (|U| + |I|) \times d \). We further define that, when the recommendation is \( \phi \), i.e., the recommendation is hidden from \( u \), the benefit function is reduced to zero. In summary, the benefit function can be formally defined as following:

\[
b_u(i) = V^T_u W_i I_{i \in I} + 0 I_{i = \phi},
\]

2) Cost Function: Cost is what one can obtain in addition from the best alternative if chosen in economics. Ideally, the cost function \( c_u(i) \) should be defined for all items. However, it is hard to model the benefit of best alternative, which is unknown. In many scenarios, a user’s cost can be well estimated independent of items. For example, cost can be estimated by users’ quality requirement for recommendations. A user who is very selective on recommendations would have high cost, as he/she is easy to be upset by bad recommendation results. Another similar estimation of cost would be the value of users’ attention of a time unit. If a user is very selective, then the value of his/her attention of a time unit is high, as limited time would be spent on the recommender system. Also, the tighter schedule users have, the higher cost they need to pay for activities on recommendation service, including viewing and making use of recommendation results. Hence, to avoid adding more complexity to our model, cost is defined as a user-dependent function, independent of items shown.

![Fig. 1: Existing recommender would generally follow the workflow that first train a model, and then make recommendations. The BC-model would adjust the ranking score returned by the underlying model and determine if the recommended items should be shown or not.](image)

Based on the analysis above, the personalized cost function \( c_u(i) \in \mathbb{R} \) is defined as:

\[
c_u(i) = C_u I_{i \in I} + 0 I_{i = \phi},
\]

where \( c_u(i) = C_u \) as the cost of each user need to pay if they choose to interact with the recommendation service.

To summarize that the ranking list for user \( u \) is based on the \( g_u(i) \) scores, where \( i \)'s are the items to be recommended, and we have \( g_u(i) = s_u(i) + V^T_u W_i - c_u(i) \), where \( s_u(i) \) is the ranking score returned by existing recommender system, while \( V^T_u, W_i, c_u(i) \) are parameters to be learned from user’s behavior data. The learning algorithm would be introduced in the next section.

### IV. Learning Algorithm

In this section, we will first present an overview of our benefit-cost model, and explain why it would work to capture users’ preferences that are missed by the underlying recommender algorithm. Then, we propose an efficient algorithm to estimate the benefit and the cost function based on Bayesian inference.

#### A. Overview

Figure 1 presents the work flow for classical recommender systems, and downside management with Benefit-Cost model (BC-Model). The BC-model is very simple and does not make any assumptions about the underlying recommender system. A natural question is: why BC-Model can achieve better performance in managing the downside of recommender systems?

The reason why BC-Model is powerful is that BC-Model learns from the user-item pairs for which underlying recommender system makes wrong prediction. BC-Model only learns from users’ interactions to items that have been recommended, and does not consider the items that have never been shown to users. All the recommended items are predicted to get positive feedbacks, however the recommender system makes mistakes sometimes (downside management). The idea is similar to Adaboost \cite{15} in which we learn an other learner based on those instances misclassified by the underlying recommender system, with only one iteration. Besides, if the underlying recommender system applies a
different model, or works with different set of features, the integration of underlying recommender system and BC-Model is a hybrid recommender system, leading it to better prediction performance.

**B. Partial Preferences**

As mentioned in (2), the final prediction is based on both the underlying recommender system and BC-Model that $g_u(i) = s_u(i) + f_u(i)$, where $f_u(i)$ is returned by underlying recommender system, and $s_u(i)$ is to be estimated.

Our goal for recommendation is to give personalized ranking of items based on overall utility of user-item pairs. To achieve downside management, we also need to give out the position of null recommendation $\phi$ in the ranking. If a top-ranked item $i$ meets the requirement $g_u(i) > g_u(\phi) = 0$, then $i$ would be shown, otherwise, $i$ would be hidden. As it would be hard to model the full ranking, and the full ranking can be derived from the partial order of every two items, we consider predicting personalized partial order of any two items from $\{\phi \cup I\}$ in our model. That is, for each user $u$, we want to provide a partial order $<_u \subseteq (\{\phi \cup I\})^2$.

To ensure that $<_u$ can give out the order of any items or $\phi$ pair, $<_u$ has to satisfy the following properties [16]:

- $\forall i,j \in I : i \neq j \Rightarrow i <_u j \lor j <_u i$, (totality)
- $\forall i,j \in I : i <_u j \land j <_u i \Rightarrow i = j$, (antisymmetry)
- $\forall i,j,k \in I : i <_u j \land j <_u k \Rightarrow i <_u k$. (transitivity)

Given a user $u$, and recommendations $i, j$, if $i <_u j$, then define $(u, i, j)$ as a personalized partial preference. All personalized partial preferences can be derived from users’ behavior history based on the order of different feedback categories defined in (1):

$$\forall (u, i, j) \in R_k, \forall (u, j) \in R_{k2} : R_{k1} < R_{k2} \Rightarrow i <_u j$$

(7)

The learning framework we propose is to estimate the benefit function and cost function by maximize the likelihood of observed personalized partial preferences. We can obtain all the personalized partial preferences based on (7), i.e., the training dataset as following:

$$D := \{(u, i, j) \in U \times I \times I | (u, i, j) \in R, i <_u j\}$$

**C. Optimization**

a) Bayesian partial order: We formulate the catastrophic recommendation problem as an optimization by maximizing the likelihood of partial orders inferred from observed action logs. We use the Bayesian Personalized Ranking optimization criterion (BPR-Opt) [16]. The BPR-Opt is derived by a Bayesian analysis to estimate the probability of $P(i <_u j | \Theta)$, where $\Theta$ contains model parameters, and has prior probability $p(\Theta)$. Based on Bayesian analysis, by maximizing the following posterior probability we can obtain the best personalized ranking $<_u \subseteq I^2$ that

$$p(\Theta | <_u) \propto p(<_u | \Theta) p(\Theta)$$

(8)

$p(\Theta)$ is defined as normal distribution with zero mean, with covariance matrix $\Sigma_\Theta = \lambda_\Theta I$, where is $I$ is identity matrix.

$p(\Theta)$ can be written as $p(\Theta) \sim N(0, \Sigma_\Theta)$. In our benefit cost model, $\Theta = \{V \in \mathbb{R}^{\mid I \mid \times \mid I \mid}, W \in \mathbb{R}^{\mid I \mid \times \mid I \mid}, C \in \mathbb{R}^{\mid U \mid} \}$.

We make independent assumptions that (1) users’ behaviors are independent of each other, and (2) the order of each pair of items for a specific user $u$ is independent of each other. So that (8) can be written as the following:

$$\prod_{u \in U} P(\Theta | <_u) = \prod_{(u, i, j) \in D} p(i <_u j | \Theta).$$

(9)

The probability of observing a partial order $(u, i, j) \in D$ is defined using the logistic loss function:

$$p(i <_u j | \Theta) := \sigma(\hat{g}_{uij}(\Theta)),$$

where $\sigma$ is the logistic sigmoid, i.e., $\sigma(x) := \frac{1}{1 + e^{-x}}$, and $\hat{g}_{uij}$ is defined as the net utility distance:

$$g_{uij} = g_u(i) - g_u(j).$$

(11)

Substitute (5), (6) into (11) and (9), we can obtain the optimization objective as following:

$$O_{PT} = \ln p(\Theta | <_u) = \ln p(\Theta | <_u) p(\Theta) = \sum_{(u, i, j) \in D} \ln \sigma(\hat{g}_{uij}(\Theta)) + \ln p(\Theta)$$

D. Parameters Inference

As the optimization criterion $O_{PT}$ defined above is not concave, we propose to use alternative maximization with accelerated gradient descent to solve this problem.

1) Fixing the latent feature values of users $V$, update the latent feature values of items $W$ and users’ cost $C$ using accelerated gradient descent. The gradient descent $\frac{\partial O_{PT}}{\partial W_{ik}}$ and $\frac{\partial O_{PT}}{\partial C_u}$ given $V$ are as following:

$$\frac{\partial O_{PT}}{\partial W_{ik}} |_V = \sum_{(u, i, j) \in D} \frac{e^{-\hat{g}_{uij} + \hat{g}_{uij}}}{1 + e^{-\hat{g}_{uij} + \hat{g}_{uij}}} V_{uk} - 2\lambda_\Theta W_{ik}$$

Similarly,

$$\frac{\partial O_{PT}}{\partial C_u} |_V = \sum_{(u, i, j) \in D} \frac{e^{-\hat{g}_{uij} + \hat{g}_{uij}}}{1 + e^{-\hat{g}_{uij} + \hat{g}_{uij}}} \left(\frac{\partial \hat{g}_{uij}(\Theta)}{\partial C_u} - \frac{\partial \hat{g}_{uij}(\Theta)}{\partial C_u}\right) - 2\lambda_\Theta C_u$$

where $\frac{\partial \hat{g}_{uij}(\Theta)}{\partial C_u} = I \{i \in I\}$.

2) Fixing the latent feature values of items $W$, $V$ and $C$ using accelerated gradient descent. Thus, $\frac{\partial O_{PT}}{\partial V_{uk}}$ and $\frac{\partial O_{PT}}{\partial V_{uk}}$ given $W$ are as following:

$$\frac{\partial O_{PT}}{\partial V_{uk}} |_W = \sum_{(u, i, j) \in D} \frac{e^{-\hat{g}_{uij} + \hat{g}_{uij}}}{1 + e^{-\hat{g}_{uij} + \hat{g}_{uij}}} (W_{ik} - W_{jk}) - 2\lambda_\Theta V_{uk}$$

$$\frac{\partial O_{PT}}{\partial C_u} |_W = \frac{\partial O_{PT}}{\partial C_u} |_V.$$
Fig. 2: Convergence comparison between Stochastic Gradient Descent (SGD) and Alternative Maximization with Accelerated Gradient Descent (AGD). The measurement is relative Area Under Precision Recall curves.

V. EXPERIMENTS

We evaluate the proposed benefit-cost model with regard to both the upside and the downside performance, we study the effect of B-C model for both upside and downside recommendation. We also validate the effectiveness of zero threshold in helping identify the downside.

A. Datasets and baselines

1) LinkedIn Dataset: The LinkedIn dataset consists of real-world job recommendation data and the corresponding user interaction tracking data. The underlying recommendation algorithm used to produce the recommendation is content-based. The item ranking scores, \( s_{ui} \), are obtained by calculating the similarity between user and job profiles. We collect user-item interaction pairs, \((u,i)\), that are qualified to fall into any of the user behavior categories defined in Section II-B1. Then, we split the resulting interaction data into training (80%) and testing (20%) for the experiment. Statistics of this dataset can be found in Table I.

2) Yelp Phoenix Dataset: The Yelp Phoenix [13] dataset consists of user feedback towards merchants in terms of a review score ranging from 1 to 5. We select a 10-core\(^1\) subset of Yelp dataset, and split it into 3 parts (40%, 40% and 20%), corresponding to the 3 different stages shown in Figure I. The splitting criteria is based on the time of reviews. The way we identify user behavior category is based on the number of different rating scores a user has produced. For users with at least 4 different rating scores, items that receive the lowest rating score are identified as the downside, and should be ideally hidden from the users. For users with less than 4 different rating scores, the highest ones are identified as strong positive, and second highest ones are identified as weak positive, and so forth. The statistics of the Yelp dataset is shown in Table I. The underlying recommendation algorithm for Yelp dataset is Matrix Factorization [12] as there is no content information available about users and businesses.

<table>
<thead>
<tr>
<th>dataset</th>
<th>users</th>
<th>items</th>
<th>user-item</th>
</tr>
</thead>
<tbody>
<tr>
<td>LinkedIn</td>
<td>28,846</td>
<td>21,141</td>
<td>364,185</td>
</tr>
<tr>
<td>Yelp</td>
<td>2,824</td>
<td>2,679</td>
<td>103,976</td>
</tr>
</tbody>
</table>

TABLE I: Basic statistics of the datasets.

B. Baseline Methods

We use some common commendation algorithms as the baseline methods and apply the proposed benefit-cost model on top of them. We use the ranking scores of underlying recommendation system as the net utility score. Different global thresholds would be applied to get acceptable recommendations and catastrophic recommendations, within top ranked recommendations from the baseline model. Recall that the underlying recommendation model can be arbitrary. In our experiments, we use a logistic regression model based on content similarity as the baseline for the LinkedIn dataset and a Matrix Factorization recommendation algorithm from [12] for the Yelp dataset.

C. Evaluation Metrics

We will show the influence of downside management on the personalized ranking. When measuring the ranking performance of downside management of recommendation, we use Area Under Precision-Recall Curve (AUPR) [7]. The precision-recall curve is plotted by computing precision and recall at every ranking position of the returned personalized ranking list. The numbers are all relative to the baseline models to show the performance gain. We use AUPR under two different settings to evaluate the performance:

- Upside measurement. Label recommendations that receive positive feedback as positive;
- Downside measurement. Label recommendations that receive negative feedback as positive.

\(^{1}\)p-core means each user has interactions with at least \( p \) items and each item has interactions with at least \( p \) users.
When measuring the upside performance of recommendation system, we label the recommendations that receive positive feedback as positive, and use AUPR as evaluation measurement. The results are shown in Table II. For both LinkedIn and Yelp dataset, the benefit-cost model beats the underlying recommendation algorithms, that is to say incorporating downside management by benefit-cost model does not have negative impact on the upside performance, but rather improve the performance because of the more accurate estimation of the recommendation utility. The relative precision-recall curve of the Yelp dataset is shown in Figure 3a, and LinkedIn dataset is shown in Figure 3c.

**TABLE II:** AUPR gain measurement on upside management and downside management. By setting the AUPR of baselines to be 100%, we obtain the AUPR gain of BC-model.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Method</th>
<th>Upside AUPR</th>
<th>Downside AUPR</th>
</tr>
</thead>
<tbody>
<tr>
<td>LinkedIn</td>
<td>BC-Model</td>
<td>112.1%</td>
<td>138.1%</td>
</tr>
<tr>
<td></td>
<td>Similarity</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Yelp</td>
<td>BC-Model</td>
<td>105.7%</td>
<td>124.40%</td>
</tr>
<tr>
<td></td>
<td>MF</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

**E. Results of the Downside Performance**

For measurement of downside performance, we measure the results from two different perspectives, ranking and classification. That is to say, we are interested in learning if the benefit-cost model is able to rank the bad recommendations better than the existing algorithm, as well as if the benefit-cost model can derive an accurate threshold to determine if the recommendation should be shown or not (Section V-F).

When evaluating the downside performance, bad recommendations from the worst user behavior category are labeled as positive examples. We then rank the items in decreasing order of the overall utility scores. The experimental results are shown in Table II in terms of the AUPR. We can see that the benefit-cost model outperforms in the downside by a big margin. The relative precision-recall curve of Yelp dataset is shown in Figure 3b, and LinkedIn dataset in Figure 3d.

**F. Crowdsourcing**

For better understanding of the performance of our downside management model, we conduct a crowdsourcing evaluation task in the context of job recommendation. The training data is collected from users’ activity tracking data and the baseline recommendation model is based on content similarity. We interview a group of participants about their feedback on top ranked recommendations returned by the baseline model. Among these recommendations, there are 19 perceivably unacceptable ones which become the gold set in evaluation. Applying our benefit-cost model on top of the baseline model, we find 25 recommendations received negative predicted net utility, among which 13 are correct according to the gold set. So the precision is 0.52, while recall is 0.684. If we use the baseline algorithm alone, both the precision and the recall are 0, i.e., none of these bad recommendations can be detected.

**G. The Effect of Different Latent Dimensions**

In modeling the benefit, different dimensions can be chosen. We want to study how dimensions influence both the upside and the downside performance. We test the model with different dimensions, and the results are shown in Figure 4. In both cases, the performance gain is little or even negative by increasing the dimension when $d > 64$. 

**VI. RELATED WORK**

There has been a plethora of research in the field of recommender systems. For a recent survey, we refer readers to Bobadilla et al. [3]. In general, there are three types of filtering used in recommender systems: content filtering, demographic, and collaborative filtering [14], [19]. Content filtering typically compares the profile similarity between users and items. Demographic filtering makes recommendation based on users’ demographic information, and then derives recommendations by incorporating some heuristics or rules learned from the user-item interaction history. Collaborative
filtering (CF), including memory-based, model-based, and hybrid algorithms, is one of the most successful family of methods for recommender systems.

The task of downside management in recommender systems can be viewed as to predict the difficulty of finding undesired recommendations for a user, which is quite similar to predict the query difficulty in the search community [8], [1]. Estimating the query difficulty is to quantify the quality of search results retrieved for a query from a given collection of documents, so that search engines can handle the "difficult" queries properly [5]. In [6], Cronen-Townsend et al. propose to calculate the clarity score, which measures the coherence of the language usage in documents and queries, to predict query difficulty. However, when there is no relevance information available for the queries, there is no systematic way to define a threshold for clarity score. In [10], He et al. propose three query coherence scores: simple coherence, coherence with global constraint, and coherence with proximity constraint to capture the semantic similarity among query topic aspects, and then predict query difficulties. Unlike [6], [10], which only make use of the features of queries, Guo et al. [8] study various features including features of the query, search results, and user interaction with the search results to predict query performance. However, these work of predicting query difficulty cannot be simply applied to a recommender system as a users' profile is substantially longer than a query. It is worth noting that, instead of deciding to show or not to show a recommendation, in [21], [20] a new problem is proposed for determining when the best time is to make recommendations. In [21], Wang et al. adopt a proportional hazards model to capture the tenure between two successive decisions and related factors, and then extend the model with a hierarchical Bayesian framework.

In the proposed learning framework, we use the ranking scores of underlying recommendation systems and users' historical action logs as training data, and estimate benefit and cost in the residual model. Residual model is proposed by Koren in [11]. However, residual model is used to directly model ratings in [11], while we use residual model to estimate the difference between ranking score of an underlying system and users’ actual unobserved utility from interacting with recommendations.

VII. CONCLUSION

In this work, we identify the problem of the downside management in recommender systems. The goal is to detect and hide individual undesirable recommendations, especially for cases where the best recommendations available from the existing recommendation algorithm are actually bad ones. We demonstrate that by detecting and hiding bad recommendations, a Pareto improvement can be achieved in terms of the overall utility for users and content providers. Based on the users’ behavior categorization together with a residual model based on the benefit-cost analysis, we formulate the problem as an optimization problem and develop an efficient learning algorithm. The resulting downside management method runs on top of existing recommender systems, which makes it a very versatile method. Experiment results on large real-world datasets demonstrate that the proposed method can effectively achieve improvement in both the upside and, especially, the downside performance.

REFERENCES