Causal Collaborative Filtering

Shuyuan Xu
Rutgers University
New Brunswick, NJ, US
shuyuan.xu@rutgers.edu

Yingqiang Ge
Rutgers University
New Brunswick, NJ, US
yingqiang.ge@rutgers.edu

Yunqi Li
Rutgers University
New Brunswick, NJ, US
yunqi.li@rutgers.edu

Zuohui Fu
Rutgers University
New Brunswick, NJ, US
zuohui.fu@rutgers.edu

Xu Chen
Renmin University of China
Beijing, China
xu.chen@ruc.edu.cn

Yongfeng Zhang
Rutgers University
New Brunswick, NJ, US
yongfeng.zhang@rutgers.edu

ABSTRACT

Recommender systems are important and valuable tools for many personalized services. Collaborative Filtering (CF) algorithms—among others—are fundamental algorithms driving the underlying mechanism of personalized recommendation. Many of the traditional CF algorithms are designed based on the fundamental idea of mining or learning correlative patterns from data for matching, including memory-based methods such as user/item-based CF as well as learning-based methods such as matrix factorization and deep learning models. However, advancing from correlative learning to causal learning is an important problem, because causal/counterfactual modeling can help us to think outside of the observational data for user modeling and personalization. In this paper, we propose Causal Collaborative Filtering (CCF)—a general framework for modeling causality in collaborative filtering and recommendation. We first provide a unified causal view of CF and mathematically show that many of the traditional CF algorithms are actually special cases of CCF under simplified causal graphs. We then propose a conditional intervention approach for *do*-calculus so that we can estimate the causal relations based on observational data. Finally, we further propose a general counterfactual constrained learning framework for estimating the user-item preferences. Experiments are conducted on two types of real-world datasets—traditional and randomized trial data—and results show that our framework can improve the recommendation performance of many CF algorithms.

KEYWORDS

Collaborative Filtering; Causal Analysis; Causal Learning; Intervention; Counterfactual Reasoning; Recommender Systems

1 INTRODUCTION

Recommender systems are important and valuable tools for many Web-based services such as e-commerce, social networks and online media systems. Collaborative Filtering (CF) [8, 37] algorithms, among others, are fundamental algorithms that support the underlying mechanism of recommender systems.
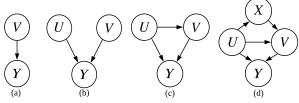
Figure 1: Many traditional CF models are special cases of CCF under simplified causal graphs. In the graphs, U is user, V is item, X is user interaction history, Y is preference score. (a) Causal graph for non-personalized models. (b) Causal graph for similarity matching-based CF models. (c) Causal graph that considers the causality from user to item [4]. (d) Causal graph used in our framework to demonstrate the idea of CCF, using user interaction history X as a mediator.

In this paper, we propose Causal Collaborative Filtering (CCF). Different from traditional collaborative filtering algorithms whose ultimate goal is to estimate the correlation $P(y|u, v)$, casual collaborative filtering aims to estimate the causal relation $P(y|u, do(v))$, where u, v is a user-item pair and y is the preference score to be estimated for the pair, e.g., $y = 1$ for likes and $y = 0$ for dislikes. Here, *do*-calculus is used to represent the causal effect if we intervene to recommend item v instead of passively observing item v in training data. More interestingly, we show that traditional CF models are actually special cases of CCF under simplified causal graphs (Figure 1, more details later), and CCF is a general framework for casual learning in recommender systems which can be applied over various causal graphs. We finally propose a counterfactual constrained learning approach to estimate the causal relation $P(y|u, do(v))$.

More specifically, traditional CF-based models are typically framed as predicting users' preference scores over items. Mathematically, this can be expressed as predicting the $P(y|u, v)$. For example, non-personalized popularity-based algorithms [14] assume $P(y|u, v) \propto P(y|v)$ and rank the items according to item popularity; user-based CF [19, 34] assumes $P(y|u, v) \propto \frac{1}{|N(u)|} \sum_{u' \in N(u)} y_{u'v}$ where $N(u)$ are user u's neighbours and $y_{u'v}$ are neighbours' ratings on item v; item-based CF [24, 36] assumes $P(y|u, v) \propto \frac{1}{|N(v)|} \sum_{v' \in N(v)} y_{uv'}$ where $N(v)$ are item v's neighbours and $y_{uv'}$ are neighbours' ratings from user u; Matrix Factorization (MF) models such as [20] assumes $P(y|u, v) \propto \mathbf{u}^\mathsf{T}\mathbf{v}$ or $\mathbf{u}^\mathsf{T}\mathbf{v} + b_u + b_v + b$, where $\mathbf{u}$ and $\mathbf{v}$ are user/item latent factors and b_* are bias terms; Probabilistic MF such as [28] assumes $P(y|u, v) \propto \mathcal{N}(y|\mathbf{u}^\mathsf{T}\mathbf{v}, \sigma^2)$, where $\mathcal{N}$ is a normal distribution, and recent deep models assume $P(y|u, v) \propto \text{NN}(\mathbf{u}, \mathbf{v})$,

where NN is a neural network for similarity matching. From a general perspective, these models are all trying to estimate the $P(y|u, v)$ under different modeling assumptions.

However, the essential goal of recommendation is not only to estimate the pre-intervention user-item associative relationships, but to estimate the post-intervention effects if we recommend/display something to users. This can be interpreted as trying to answer the "what if" question: what would happen if we intervene to recommend a certain item to a target user.

Using standard mathematical language of causal inference [30], the above "what if" question can be represented as $P(y|u, do(v))$, where the do-operation is used to model the interventions. As a result, we propose Causal Collaborative Filtering (CCF) in this paper, which aims to estimate $P(y|u, do(v))$ instead of $P(y|u, v)$ for personalized recommendation. Interestingly, we find that traditional CF models are actually special cases of CCF, i.e., they are actually also trying to estimate $P(y|u, do(v))$—but under simplified (sometimes unrealistic) causal graphs. As a result, CCF mathematically subsumes many of the traditional CF models. For example, non-personalized models such as popularity ranking assume the causal graph in Figure 1(a). Since user is excluded from the graph and item is a root node in the graph, we have $P(y|u, do(v)) = P(y|do(v)) = P(y|v)$, which naturally reduces to the estimate of popularity ranking. Many similarity matching-based CF models such as user/item-based CF, matrix factorization and neural ranking models assume the collider casual graph in Figure 1(b), where the user and item are assumed to appear independently in the observational training data (more details explained in Section 4). Since both U and V are root nodes in the graph, we have $P(y|u, do(v)) = P(y|u, v)$, which also reduces to the associative estimates of these models. Except for the above associative matching-based models, some existing causal recommendation models can also be included in the $P(y|u, do(v))$ framework. For example, Causal Embeddings for Recommendation (CausE) [4] assumes the causal graph in Figure 1(c), which removes the independence assumption between user and item, and adopts direct intervention through randomized treatment to estimate $P(y|u, do(v))$, as will be explained in Section 4.

Except for the above conceptual contribution, our work also provides technical contributions. More specifically, a great challenge is how to estimate $P(y|u, do(v))$. Estimating $P(y|u, do(v))$ using direct intervention requires access to the recommendation platform so that we can deploy the intervention strategies such as displaying randomized recommendations to users. However, such platform is hardly accessible to researchers. Even in industry environments where the platform is fully accessible, researchers and developers are usually prevented from large-scale randomized trails because that will greatly hurt the user experience. In this work, we propose a conditional intervention approach to estimating $P(y|u, do(v))$ based on observational data. Specifically, we adopt the causal graph in Figure 1(d) for conditional intervention, which considers the user interaction history X for mediator analysis. Moreover, solving the conditional intervention requires counterfactual reasoning, and we further propose a counterfactual constrained machine learning approach for counterfactual reasoning in both discrete and continuous space. More details will be explained in Section 5. Finally, we conduct extensive experiments on two different types

of real-world datasets: type-one datasets are traditional training-validation-testing split datasets, while for the type-two datasets, the testing set is collected through randomized trials. Experimental results show that our CCF framework significantly improves the recommendation performance on both types of datasets.

In the following parts of the paper, we will introduce the related work in Section 2 as well as the notations and preliminaries in Section 3. In Section 4, we will provide a unified causal view of Collaborative Filtering (CF) and motivate the basic idea of Causal Collaborative Filtering (CCF). Then, we will introduce the proposed CCF framework in Section 5 and provide the experimental results in Section 6. Finally, Section 7 concludes the work and discusses about the potential future research directions.

2 RELATED WORK

Collaborative Filtering (CF) [8] is one of the most fundamental approaches to personalized recommendation, which has been widely used in many real-world systems. The key idea behind many of the CF models is that similar users may share similar interests and similar items may be liked by similar users. Due to the wide scope of literature of CF, it is hardly possible to cover all of the CF algorithms, so we review some representative methods in this section, and a more comprehensive review can be seen in [8, 49].

Early memory-based CF models—such as user-based CF [19, 34] and item-based CF [24, 36]—take the row or column vectors of the user-item rating matrix as the user and item vector representations, and calculate the similarity between users or items for recommendation based on pre-defined similarity functions such as cosine similarity and Pearson correlation coefficient. To extract latent semantic meanings from the matrix, researchers later explored learned user and item vector representations. This started with Latent Factor Models (LFM) such as matrix factorization [20], probabilistic matrix factorization [28], tensor factorization [18] and factorization machines [31], which are widely adopted models in practice. In these models, each user and item is learned as a latent representation to calculate the matching score of each user-item pair, usually based on inner-product.

The development of deep learning and neural networks has further extended CF. The relevant methods can be broadly classified into two categories: similarity learning approach and representation learning approach. The similarity learning approach adopts simple user/item representations (such as one-hot) and learns a complex matching function (such as a prediction network) to calculate user-item matching scores [7, 10, 13, 43], while the representation learning approach learns rich user/item representations and adopts a simple matching function (e.g., inner product) for efficient matching score calculation [2, 27, 47, 50, 52]. User representations can also be directly calculated from the user's interaction histories, such as in sequential recommendation [6, 11, 17, 33, 40]. Another important direction is learning to rank for recommendation, which learns the relative ordering of items instead of the absolute scores. A representative method on this direction is Bayesian Personalized Ranking (BPR) [32], which is a pair-wise learning to rank method.

Most of the existing methods learn correlative patterns from data for matching and recommendation based on either simple or complex matching functions. However, advancing from correlative

learning to causal learning is an important problem [30]. The community has explored causal modeling on several different perspectives. For example, researchers adopted causal models to generate explanations for recommendation [9], explored fairness issues under counterfactual settings [21], and corrected data bias for ranking in search [16], recommendation [4, 25, 38], advertising [46] and evaluating the ranking models [44]. Many of the models are based on Inverse Propensity Score (IPS) methods, which aim to turn the outcomes of an observational study into pseudo-randomized trials by re-weighting the samples. Though convenient in implementation, the disadvantage is that the estimator may not properly handle large shifts in observational probability [4]. Besides, each model is usually specifically designed for a particular scenario or a particular problem setting, while there still lacks a general framework for modeling counterfactual reasoning in collaborative filtering, which is the fundamental problem we aim to solve in this work.

3 NOTATIONS AND PRELIMINARIES

In this section, we introduce the basic notations used throughout the paper. We also introduce some fundamental concepts of causal inference to be used in the following parts of the paper.

3.1 Basic Notations

In this paper, we use uppercase letters such as Y to represent random variables. In particular, we use U, V, X, Y to represent user, item, history, and preference variables. We use lowercase letters such as y to represent the specific value that a random variable can take. In particular, we use u, v, x, y to represent a specific user, item, history, and preference value. Moreover, we use bold font lowercase to represent the latent vector embedding of users and items, such as $\mathbf{u}, \mathbf{v} \in \mathbb{R}^D$, where D is the dimension of the embedding vectors.

We use probability notations such as $P(Y = y)$ and $P(Y = y|X = x)$ to represent the probability or the conditional probability that a variable Y takes on a specific value y. When the context is clear, the probability is simplified as $P(y)$ and $P(y|x)$. In this work, for simplicity, we consider binary values for the user-item preference variable Y, i.e., Y can take on two values 1 and 0, denoted as $P(Y = 1)$ and $P(Y = 0)$, corresponding to the probability of *like* or *dislike* an item, respectively. However, the framework can be generalized to multiple value cases.

We adopt standard intervention and counterfactual notations as in Pearl et al. [30]. In particular, $P(Y = y|do(X = x))$—simplified as $P(y|do(x))$ when the context is clear—denotes the probability of $Y = y$ under intervention $X = x$. Moreover, $P(Y = y|do(X = x), Z = z)$—simplified as $P(y|do(x), z)$—denotes the z-specific intervention. We use $P(Y_{X=x'} = y|U = u, X = x)$—simplified as $P(Y_{x'} = y|u, x)$ or $P(Y_{x'}(u) = y|x)$—to denote the probability of $Y = y$ for individual $U = u$ in the counterfactual world where $X = x'$, given that what happened in the real world is $X = x$. When calculating expectation is needed, we use $E[Y_{x'}(u)|x]$ to represent the expected value of Y in the counterfactual world.

3.2 Basic Concepts in Causal Inference

DEFINITION 1. *(Structural Causal Models) [30, p.26] A structural causal model (SCM) M consists of two set of variables U and V, and a set of functions f that assign a value to each variable in V based on other variables in the model. Here U are exogenous variables that no explanatory mechanism is encoded.*

DEFINITION 2. *(Causal Graph) [30, p.35] A causal graph is a directed acyclic graph (DAG) $\mathcal{G} = (\{U, V\}, E)$, which captures the relationships among the variables in the corresponding SCM.*

DEFINITION 3. *(Intervention) [30, p.55] We distinguish between cases where a variable X takes a value x naturally and cases where we fix X = x by denoting the later do(X = x). So $P(Y = y|X = x)$ is the probability that Y = y conditioned on finding X = x, while $P(Y = y|do(X = x))$ is the probability that Y = y when we intervene to make X = x. Similarly, we write $P(Y = y|do(X = x), Z = z)$ to denote the conditional probability of Y = y, given Z = z, in the distribution created by the intervention do(X = x).*

To calculate the causal effect $P(y|do(x))$, the most fundamentalist approach is through causal graph manipulation. Technically, we remove all of X's incoming edges from the original causal graph $\mathcal{G}$ to create the manipulated graph $\mathcal{G}_m$. And then we have have $P(y|do(x)) = P_m(y|x)$, where P_m is the manipulated probability. However, really implementing the manipulated causal graph $\mathcal{G}_m$ through direct intervention to calculate P_m can be challenging or even impossible in practice. As a result, it would be nice if we can estimate $P(y|do(x))$ from purely observational data. The following causal effect rule answers this question.

DEFINITION 4. *(The Causal Effect Rule) [30, p.59] Given a causal graph $\mathcal{G}$ in which a set of variables PA are designated as the parents of X, the causal effect of X on Y is given by:*

$$
\begin{aligned}
P(Y = y|do(X = x)) &= \sum_z P(Y = y|X = x, PA = z)P(PA = z) \\
&= \sum_z \frac{P(X = x, Y = y, PA = z)}{P(X = x|PA = z)}
\end{aligned}
\tag{1}
$$

where z ranges over all the combinations of values that the variables in PA can take. The factor $P(X = x|PA = z)$ is the "propensity score".

The most important benefit brought by the above rule is that it enables us to calculate the causal effect between two variables based on passive observational data—we see that the right side of the equation does not include *do*-calculations any more. However, enumerating all parents' value combinations is still rather complicated. The following backdoor criterion solves the problem.

DEFINITION 5. *(Backdoor Criterion) [30, p.61] A set of variables Z satisfies the backdoor criterion related to an ordered pair of variables (X, Y) in a causal graph $\mathcal{G}$ if Z satisfies both (1) No node in Z is a descendant of X and (2) Z blocks every path between X and Y that contains an arrow into X.*

If a set of variables Z satisfies the backdoor criterion for X and Y, then the causal effect of X on Y is given by the formula:

$$
P(Y = y|do(X = x)) = \sum_z P(Y = y|X = x, Z = z)P(Z = z) \tag{2}
$$

through which we can also estimate $P(y|do(x))$ from observational data but it may only need much fewer variables.

The backdoor criterion can be generalized to z-specific causal effect $P(Y = y|do(X = x), Z = z)$, in which we care about the causal effect of X on Y under a specific value $Z = z$ [30, p.70]. If we

can find a set of variables S such that $S \cup Z$ satisfies the backdoor criterion (S may include Z), then the z-specific causal effect can be estimated from observational data:

$$P(Y = y | do(X = x), Z = z)$$
$$= \sum_s P(Y = y | X = x, S = s, Z = z) P(S = s | Z = z) \quad (3)$$

where the summation goes over all value combinations of S.

Finally, counterfactual analysis aims to answer queries that go beyond the observational data. In notation, we use $Y_{X=x}(U = u) = y$, or simplified as $Y_x(u) = y$, to represent the counterfactual sentence "Y would be y had X been x, in situation $U = u$," though the observed value of X in real world is not x. The mathematical definition of counterfactual is as follows.

DEFINITION 6. *(Counterfactual) [30, p.94] Let M be the original structural causal model, and M_x be the modified version of M with the equation of X replaced by $X = x$. Then the formal definition of the counterfactual $Y_x(u)$ is $Y_x(u) = Y_{M_x}(u)$, i.e., the counterfactual $Y_x(u)$ in model M is defined as the solution for Y in the "surgically modified" submodel M_x.*

Further more, we use $P(Y_x = y)$ to represent the probability of $Y = y$ had X been x, and use $E[Y_x]$ to represent the expected value of Y had X been x.

4 A UNIFIED CAUSAL VIEW OF CF

We start by providing a unified causal view of collaborative filtering (CF). Specifically, we show that the fundamental goal of many CF algorithms for personalized recommendation is to estimate the causal effect $P(Y = y | U = u, do(V = v))$, simplified as $P(y | u, do(v))$, where U, V, Y represent the user, item and the user's preference on the item, respectively. According to the definition of causal effect [30, p.55], $P(y | u, do(v))$ represents the causal effect of item v on the preference y conditioned on user u, which is the probability that user u's preference score is y (e.g., 1 or 0) when we *intervene* to recommend item v. The key difference between various CF models is that they assume different causal graphs to calculate $P(y | u, do(v))$. When the causal graph is too simple or even unrealistic, the causal effect will naturally degenerate to association relations that are considered in traditional CF models.

Before we proceed to formally compare different CF models, we first explain the intuition of $P(y | u, do(v))$, which can be appreciated from both recommendation perspectives and causal perspectives. On the recommendation perspective, we usually consider *personalized* recommendations in modern recommender systems, and thus u appears in the condition to enable conditional causal effect. From the causal perspective, simply estimating $P(y | u, v)$ from observational data (as many previous models did) can only extract associative signals from the training dataset, which may be subject to the Simpson's paradox [30] since a user's personalized preference can be overwhelmed by the population effect. In contrast, $P(y | u, do(v))$ aims to estimate the causal effect if we *intervene* to recommend item v for a specific user, and the causal effect is expected to reveal the user's real preference on the item without being influenced by other confounding factors.

We now show how different CF models fit into the unified causal view under $P(y | u, do(v))$.

4.1 Non-Personalized Model

Non-personalized recommendation models such as most popular recommendation [14] assumes a simple causal graph without the user node, as shown in Figure 1(a). Since user is excluded from consideration and since item is a root node in the graph, we have $P(y | u, do(v)) = P(y | do(v)) = P(y | v)$, and $P(y | v)$ naturally represents the popularity of item v in the data.

4.2 Associative Matching Model

Most CF algorithms fall into the user-item associative matching category, such as user-based [19, 34] or item-based [24, 36] CF, matrix factorization models [20, 28], as well as many neural network-based matching models, including both function learning [10] and representation learning [50] approaches. These models actually assume a causal graph shown in Figure 1(b), where user node U and item node V constitute a collider to influence preference node Y. Basically, these models assume that the appearance of users and items are independent from each other in observational data (though this may be an unrealistic assumption), and since both U and V are root nodes, we have $P(y | u, do(v)) = P(y | u, v)$, which can thus be estimated from observational data using various models. This is actually what we have seen in many CF models for years.

The main difference of various models is how to design the matching function to estimate $P(y | u, v)$. For example, user-based CF assumes $P(Y = 1 | u, v) \propto \frac{1}{|N(u)|} \sum_{u' \in N(u)} y_{u', v}$, while item-based CF assumes $P(Y = 1 | u, v) \propto \frac{1}{|N(v)|} \sum_{v' \in N(v)} y_{u, v'}$, where $N(u)$ and $N(v)$ are the neighbours of user u and item v, respectively. Matrix factorization (MF) models such as [20] assumes $P(Y = 1 | u, v) \propto \mathbf{u}^\top \mathbf{v}$ or $\propto \mathbf{u}^\top \mathbf{v} + b_u + b_v + b$, while probabilistic matrix factorization such as [28] assumes $P(y | u, v) \propto \mathcal{N}(y | \mathbf{u}^\top \mathbf{v}, \sigma^2)$, where $\mathcal{N}$ is a normal distribution with $\mathbf{u}^\top \mathbf{v}$ as mean. Some neural network-based models such as [10, 43] assume $P(Y = 1 | u, v) \propto NN(\mathbf{u}, \mathbf{v})$, where NN is a neural network for similarity matching, while representation learning-based models such as [50] assumes $P(Y = 1 | u, v) \propto \mathbf{NN}(u)^\top \mathbf{NN}(v)$, where $\mathbf{NN}$ is a network that learns the user and item vector representations, and a simple inner product is used for similarity matching. More complex deep representation learning models such as sequential models [6, 12, 22, 40] and graph-based models [2, 41, 45, 48] can be represented as $P(Y = 1 | u, v) \propto NN(\mathbf{NN}(u), \mathbf{NN}(v))$, where a neural similarity network NN is applied on top of the neural representation learning network $\mathbf{NN}$.

Though different models design different methods to estimate $P(y | u, v)$, a fundamental assumption shared by these models is that the co-occurrence of user and item is independent in observational data, which is implied by the causal graph in Figure 1(b), and thus $P(y | u, do(v))$ can be estimated from observational data as $P(y | u, v)$.

However, this assumption is unrealistic because user behavior is influenced by the recommender system for various reasons. For example, users will more likely interact with those recommended items (exposure bias [4]), items ranked at top positions (position bias [42]), or items that are more popular (popularity bias [1]). Even the modeling assumptions made in the recommendation algorithm itself may influence what users can see and interact with (inductive bias [3]). As a result, $P(y | u, do(v))$ cannot be simplified as $P(y | u, v)$, and we need to intervene on items so as to estimate the real causal effect $P(y | u, do(v))$ between an item and the user preference.

4.3 Interventional Reasoning Model

The u-specific causal effect $P(y|u, do(v))$ by definition requires interventional reasoning. If we have complete control of the recommendation platform, then we can conduct direct intervention based on randomized experiments for estimating $P(y|u, do(v))$. However, researchers usually only have a benchmark dataset and do not have access to the real-world recommendation platform. Even in industry settings, researchers are usually not allowed to make random recommendations or can only do randomized experiment on a small subset of users such as in A/B testing. Depending on if or not we have complete control of the recommendation platform, we have the following two approaches to estimate $P(y|u, do(v))$.

4.3.1 *Direct Intervention Models.*
If we have complete control of the recommendation platform or have access to a randomized treatment dataset where user is randomly exposed to items, then the straightforward way of estimating $P(y|u, do(v))$ is through direct intervention, as researchers have done in [4]. Basically, the assumed causal graph is Figure 1(c), which extends Figure 1(b) by removing the independence assumption between user and item.

We refine Figure 1(c) as Figure 2(a) to show the structural equations $V = g(U)$ and $Y = f(U, V)$, which represent the two steps of the recommendation pipeline. $V = g(U)$ represents the de facto recommendation model in the system that decides what items are exposed to the user, and $Y = f(U, V)$ represents the user's preference on the exposed item. To estimate $P(y|u, do(v))$, we resort to the most original definition of intervention through manipulated causal graphs [30, p.54]. We remove all edges directed into V in Figure 2(a)—in this case, the edge from U to V—and we have the manipulated causal graph in Figure 2(b). We thus have $P(y|u, do(v)) = P_m(y|u, v)$, where P_m is the probability distribution according to the manipulated causal graph. To estimate $P_m(y|u, v)$, we can apply a randomized exposure policy by showing random items to users thus to implement the independence between U and V. This treatment will help us to collect an unbiased dataset to estimate $P_m(y|u, v)$. More details can be seen in [4].

4.3.2 *Inverse Propensity Scoring (IPS) Models.*
In many cases, we do not have complete access to the recommendation platform. The basic idea of propensity scoring methods is to turn the outcomes of an observational study into pseudo-randomized trials by re-weighting the samples [4], so that $P(y|u, do(v))$ can be estimated from the observational data [15, 23, 38].

More formally, according to the recommendation pipeline shown in Figure 2(a), the observed user preference r_{uv} is considered as $r_{uv} \propto P(y|u, do(v))P(v|u)$, which is the multiplication between the user's real preference $P(y|u, do(v))$ and the probability that user u had a chance to see the item $P(v|u)$. As a result, we have $P(y|u, do(v)) \propto \frac{r_{uv}}{P(v|u)}$. This means that each example in the observational data should boost its probability by a factor equal to $1/P(v|u)$, and hence the name *inverse propensity scoring* (aka *inverse probability weighting* [30, p.73]), which corrects the observational data by removing the exposure bias. In this way, we can estimate $P(y|u, do(v))$ from the corrected observational data.

The advantage of IPS-based estimator is its convenience in implementation, and the disadvantage is that the estimator may not properly handle large shifts in exposure probability. For example,
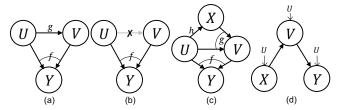

Figure 2: (a-b) Causal graphs before and after manipulation. (c-d) Reorganize causal graph using U as exogenous variable.

items with low probability of exposure $P(v|u)$ under the recommendation model will tend to have higher predicted scores [4].

4.4 Counterfactual Reasoning Model

As noted before, to estimate $P(y|u, do(v))$, direct invention approaches may not be available to us, while IPS-based approaches may be vulnerable to small probabilities. As a result, we need better approaches to the estimation. It is good to notice that $P(y|u, do(v))$ naturally involves counterfactual reasoning as part of its logic.

The reason is that most recommendation models make recommendations based on users' interaction history, including both sequential and non-sequential recommendation models. For sequential models, the algorithm naturally takes the user's interaction history as input when making recommendations. For non-sequential models such as matrix factorization, the model usually employs a matching function over the user and item embeddings to make recommendations. Even though the final recommendation step does not explicitly take user interaction history as input, the learning of user and item embeddings are based on user interaction histories. On the other hand, the user interaction history is influenced by the recommendation model itself, because users will more likely interact with the recommended items. However, using user's interaction history for recommendation is inevitable since this is the key idea of CF and it makes *personalized* recommendation possible, but because of this, it also encodes bias into the recommendation model since the user interaction history is already influenced by various factors and thus may not reveal users' real preference.

To solve the problem, we need to answer counterfactual questions such as *what if* an item had or had not been recommended, *what if* we intervene to recommend an item, and *what if* the user had a different interaction history. Such imaginary cases constitute the *counterfactual world*, in contrast to what happened in the *real world*. In the next section, we will show a flexible learning framework based on *counterfactual constraints* to answer the questions for causal collaborative filtering.

5 CAUSAL COLLABORATIVE FILTERING

To enable counterfactual reasoning, we extend the causal graph from Figure 2(a) to Figure 2(c) to consider user's interaction histories X. More specifically, the structural casual model includes three structural equations: (1) $X = h(U)$, which returns a user's interaction history X. In the most simple case, it can be a database retrieval operation that returns a user's interaction history from the observational data; (2) $V = g(U, X)$, which is the already deployed (but potentially biased) recommendation algorithm of the system that returns the recommended item V based on the user and the user's interaction history; (3) $Y = f(U, V)$, which is the unbiased

user preference function that reveals user's real preference Y on the item, and this is the function that we do not know but we want to estimate. As we can see, the fundamental goal is how to transform an arbitrary recommendation algorithm $V = g(U, X)$ into an unbiased user preference estimation $Y = f(U, V)$.

We should acknowledge that the proposed causal graph in Figure 2(c) is not a once-and-for-all solution for recommender systems, because practical recommender systems are very complicated that involve many other factors such as user and/or item content features as well as sponsored recommendations. However, we consider the proposed causal graph in this work for two reasons: (1) the structural equation $V = g(U, X)$ is general enough to include a wide scope of recommendation algorithms, including both sequential and non-sequential methods, and (2) our proposed framework—to be described in later subsections—is flexible and can be easily generalized to more complex causal graphs such as incorporating user and item content features, which we will consider in the future.

5.1 Conditional Intervention

To estimate $P(y|u, do(v))$, we first identify that $\{U, X\}$ is a set of variables that satisfy the backdoor criterion [30, p.61] for the casual effect $V \rightarrow Y$. Since we already conditioned on U for personalization, the only variable that leads to variations in V is user interaction history X, as a result, we adopt conditional intervention [30, p.70][29, p.113] to estimate $P(y|u, do(v))$.

More specifically, the recommendation policy $V = g(U, X)$ provides recommendation V based on the user U and history X, written as $do(V = g(U, X))$. To find out the distribution of the outcome Y that results from this policy, we seek to estimate $P(Y = y|U = u, do(V = g(U, X)))$. We will show that identifying the effect of such policies is equivalent to identifying the expression for the (u, x)-specific effect $P(Y = y|U = u, X = x, do(V = v))$ [30, p.71].

$$P(y|u, do(v)) \doteq P(Y = y|U = u, do(V = g(U, X)))$$
$$\overset{1}{=} \sum_{x} P(Y = y|U = u, do(V = g(U, X)), X = x) \times$$
$$P(X = x|U = u, do(V = g(U, X)))$$
$$\overset{2}{=} \sum_{x} P(Y = y|U = u, X = x, do(V = g(u, x)))P(X = x|U = u)$$
$$\overset{3}{=} \sum_{x} P(Y = y|U = u, X = x, do(V = v))|_{v=g(u,x)} P(X = x|U = u)$$
$$\overset{4}{=} \sum_{x} P(y|u, x, v)|_{v=g(u,x)} P(x|u) = E_{x|u}[P(y|u, x, v)|_{v=g(u,x)}]$$
$$(4)$$

In the above derivation, step 1 follows from the law of total probability and the fact that variables $\{U, X\}$ satisfy the backdoor criterion [30, p.61]; step 2 follows from the fact that X occurs before V and hence any control exerted on V can have no effect on the distribution of X [30, p.71]; step 3 rewrites the equation which tells us that the causal effect of a conditional policy $do(V = g(U, X))$ can be evaluated from the expression of $P(Y = y|U = u, X = x, do(V = v))$ by substituting $g(u, x)$ for v and taking the conditional expectation over X using the conditional distribution $P(X = x|U = u)$ for personalization [30, p.71]; finally, step 4 simplifies the notation into the conditional expectation form [29, p.113].

From the last step in Eq.(4) we can see that the key difference between the causal model $P(y|u, do(v))$ and traditional matching-based models $P(y|u, v)$ is the existence of the conditional probability term $P(x|u)$ in the final step. In the final step, $P(y|u, x, v)|_{v=g(u,x)}$ stands for the preference estimation of the already deployed recommendation model $V = g(U, X)$. Traditional models only consider the real world but not the counterfactual world, as a result, the conditional probability $P(x|u) = 1$ for the observed user history x, while for unobserved history x', $P(x'|u) = 0$. In this case, we can see that the summation in the final step will only include the observed history x and thus $P(y|u, do(v))$ naturally degenerates to the original recommendation model $V = g(U, X)$.

However, just because the observed history of user u is x does not mean that the user is *destined to* interact with the items in x—the user just *happened to* interact with x, i.e., if the user had a chance to be recommended with different items x' in the counterfactual world, the user may also interact with those items, and thus the probability $P(x'|u)$ is not 0. As a result, the calculation of Eq.(4) requires counterfactual reasoning in the counterfactual world where the user history had been $X = x'$, which is beyond the observational data $X = x$.

5.2 Counterfactual Reasoning

Counterfactual reasoning enables more refined intervention at individual level [30, p.78,93]. In this work, the *individual level* refers to each specific user $U = u$ for personalization purpose. To better understand this, the causal graph in Figure 2(c) is equivalently transformed into Figure 2(d), where X, V, Y form a chain structure and U serves as the exogenous variable over other variables. As a result, counterfactual reasoning is individualized on each user.

To enable counterfactual reasoning for estimating Eq.(4), let us consider a record (u, x, v, y) in the observational data, which means that the user u's real interaction history is x, and then the system logged the user's preference on item v which is y. For example, we can consider binary preference values by using $y = 1$ for likes and $y = 0$ for dislikes, but the proposed framework can also be applied to multiple preference values. According to Eq.(4), the unbiased user preference estimation $y = f(u, v)$ is expressed as

$$y = f(u, v) \propto P(y|u, do(v))$$
$$= \sum_{\tilde{x}} P(y|u, \tilde{x}, v)|_{v=g(u,\tilde{x})} P(\tilde{x}|u) = E_{\tilde{x}|u}[P(y|u, \tilde{x}, v)|_{v=g(u,\tilde{x})}]$$
$$(5)$$

To distinguish from the single real-world history x, we use $\tilde{x}$ to represent any possible user history, including both the real history x and possible counterfactual histories x'. Eq.(5) means that the estimation of $P(y|u, do(v))$ can be achieved by correcting the original recommendation algorithm's estimation $P(y|u, x, v)|_{v=g(u,x)}$ using counterfactual histories x'. More specifically, the estimation for $P(y|u, do(v))$ is the *expected* estimation of $P(y|u, x, v)|_{v=g(u,x)}$, where the expectation is taken over all possible histories (including real and counterfactual histories) when item v is recommended.

5.2.1 Generate Counterfactual Examples. Counterfactual reasoning requires generating counterfactual examples by making minimal modifications in the current model [30, p.92]. In our problem, we need to make minimal changes to the real history x so as

Table 1: Different heuristic rules to create counterfactual examples, the corresponding counterfactual question, and some intuitive toy examples. In the toy examples, the user's real interaction history x includes items $a\ b\ c$, and items at the right side of the arrow is the counterfactual history x'. Multiple counterfactual histories can be constructed from the real history x.

Heuristic Rule	Counterfactual Question	Toy Example
Keep One (K1)	What if the user only interacted with one history item?	$a\ b\ c \rightarrow a;\ a\ b\ c \rightarrow b;\ a\ b\ c \rightarrow c$
Delete One (D1)	What if the user did not interact with one of the history items?	$a\ b\ c \rightarrow b\ c;\ a\ b\ c \rightarrow a\ c;\ a\ b\ c \rightarrow a\ b$
Replace One (R1)	What if one of the history items were different?	$a\ b\ c \rightarrow a'b\ c;\ a\ b\ c \rightarrow a\ b'c;\ a\ b\ c \rightarrow a\ b\ c'$

to generate counterfactual histories x'. We start with a heuristic-based approach for counterfactual example generation and we will generalize to a learning-based approach in the next section.

We adopt three heuristic rules to generate counterfactual histories x' by applying modifications to the real history x (Table 1). The Keep One (K1) rule only keeps one item of the user's real history, the Delete One (D1) rule removes one item from the user's real history, and the Replace One (R1) rule replaces one item of the user's real history with another item. For the R1 rule, depending on how the item is replaced, we have two variants: R1-random (R1r)—the item is replaced with a random item, and R1-nearest (R1n)—the item is replaced with its nearest neighbour based on embedding similarity. We will introduce more details in the experiments.

5.2.2 **Select Counterfactual Examples**. Consider the training example (u, x, v, y) where the user's real history is x, and we are able to generate m counterfactual histories $\{x'_1, x'_2 \cdots x'_m\}$ using one of the heuristic rules. Conditional intervention (Section 5.1) requires $v = g(u, \tilde{x})$, i.e., the same item v should be recommended by the recommendation algorithm $g(\cdot, \cdot)$ under counterfactual histories (since we are considering $do(v)$ instead of just v in the condition). However, not all of the counterfactual histories $\{x'_1, x'_2 \cdots x'_m\}$ guarantee that item v is recommended under the algorithm. As a result, we execute the recommendation algorithm $g(\cdot, \cdot)$ over each counterfactual history x'_i and obtain the top-k recommendation list $\mathcal{V}'_i = g(u, x'_i)$, where k is a hyper-parameter to be tuned (will be introduced in the experiments). If the target item $v \in \mathcal{V}'_i$, then we keep the counterfactual example (u, x'_i, v, y). Suppose n of the m counterfactual histories are eventually selected, we will have a set of counterfactual examples $\{(u, x'_i, v, y)\}_{i=1}^{n}$.

5.2.3 **Calculate the Expectation**. We then calculate $P(y|u, do(v))$ based on the real observation (u, x, v, y) and the counterfactual examples $\{(u, x'_i, v, y)\}_{i=1}^{n}$ according to Eq.(5). For simplicity, we consider $P(\tilde{x}|u)$ as a piecewise uniform distribution over the real and counterfactual histories, i.e.,

$$P(\tilde{x}|u) = \begin{cases} \alpha, & \text{when } \tilde{x} = x \\ \beta, & \text{when } \tilde{x} = x'_i,\ i \in \{1, 2 \cdots n\} \end{cases}, \alpha + n\beta = 1 \quad (6)$$

where α is the probability of the real example x, and β is the probability of each counterfactual example x'_i. Since x is already observed, we apply a higher probability to x than x'_i, i.e., $\alpha > \beta > 0$. Generalizing to even more complex distributions such as Gaussian or Gamma distribution will be considered in the future. Then we have:

$$P(y|u, do(v)) = \sum_{\tilde{x}} P(y|u, \tilde{x}, v)|_{v=g(u,\tilde{x})} P(\tilde{x}|u)$$
$$= \alpha P_g(y|u, x, v) + \beta \sum_{i=1}^{n} P_g(y|u, x'_i, v) \quad (7)$$

where we use P_g to represent the probability estimation of the base recommendation algorithm $v = g(u, x)$.

5.3 Counterfactual Constraint

As we noted before, given any base recommendation algorithm $v = g(u, x)$, our goal is to derive the preference estimator $y = f(u, v) \propto P(y|u, do(v))$. To achieve this goal, we propose a counterfactual constrained learning approach. Basically, we require the base recommendation algorithm's preference estimation $P_g(y|u, x, v)$ to be equal to our wanted estimation $P(y|u, do(v))$:

$$P_g(y|u, x, v) = P(y|u, do(v)) \quad (8)$$

Eq.(8) naturally satisfies the causal graph in Figure 2(c) because variables $\{U, X\}$ satisfy the backdoor criterion for V and Y. Based on this, we can correct the estimation of any base recommender and use the corrected estimation to generate recommendations. To realize this goal, we use Eq.(8) as a counterfactual constraint to the training objective of the base recommendation algorithm.

We already know the expression for $P(y|u, do(v))$ (i.e., Eq.(7)). Combining Eq.(8) and Eq.(7), and using $P(y|u, x, v)$ for $P_g(y|u, x, v)$ for notation simplicity, we have

$$P(y|u, x, v) = \alpha P(y|u, x, v) + \beta \sum_{i=1}^{n} P(y|u, x'_i, v)$$

$$\Leftrightarrow (1-\alpha) P(y|u, x, v) = \beta \sum_{i=1}^{n} P(y|u, x'_i, v) \quad (9)$$

$$\Leftrightarrow \sum_{i=1}^{n} P(y|u, x'_i, v) = \frac{1-\alpha}{\beta} \cdot P(y|u, x, v)$$

Since we do not have prior knowledge about the counterfactual histories, the counterfactual histories x'_i are equally important to us. As a result, a sufficient condition for Eq.(9) is to require every counterfactual example's estimation to be equal to some proportion of the real-history's estimation, i.e.,

$$P(y|u, x'_i, v) = \frac{1-\alpha}{n\beta} \cdot P(y|u, x, v),\ \forall 1 \le i \le n \quad (10)$$

According to Eq.(6), we know $\alpha + n\beta = 1$. As a result, Eq.(10) can be simplified to:

$$P(y|u, x'_i, v) = P(y|u, x, v),\ \forall 1 \le i \le n \quad (11)$$

As a result, we apply the above counterfactual constraint over each (x, x') pair. In the following, we first propose a discrete version of the counterfactual constrained learning algorithm for any base recommender, which conducts counterfactual reasoning in a discrete item space. We then generalize the discrete version to a continuous version for counterfactual reasoning in a continuous latent embedding space of the items.

5.4 Counterfactual Learning

*5.4.1 **Counterfactual Learning in Discrete Space***. Let $L(g)$ be the loss function of a base recommendation algorithm $g(u, x)$. CCF aims to learn $L(g)$ under a discrete counterfactual constraint.

$$\text{minimize } L(g)$$

$$\text{s.t. } \sum_{u \in \mathcal{U}} \sum_{v \in \mathcal{I}(u)} \sum_{x' \in C(u,v)} \left| P(y|u, x', v) - P(y|u, x, v) \right| \leq \epsilon \quad (12)$$

where $\mathcal{U}$ is the set of users, x is the real history of user u, $\mathcal{I}(u)$ is the set of interacted items in the training set (excluding items in x), $C(u, v)$ is the set of counterfactual histories of user u under the target item v (section 5.2.1 and 5.2.2), and ϵ is a threshold hyper-parameter controlling how rigorous is the constraint.

*5.4.2 **Counterfactual Learning in Continuous Space***. Many recommendation models represent users, items and histories as embedding vectors in a latent space. If a user's history x is represented as an embedding vector $\mathbf{x}$, then we can directly create latent counterfactual histories $\mathbf{x}'$ by slightly perturbing vector $\mathbf{x}$ in the latent space. This will be more efficient than creating counterfactual examples in the original discrete item space based on heuristic rules. Similarly, let $L(g)$ be the loss function of a base recommendation algorithm $g(u, x)$. CCF in continuous space aims to learn $L(g)$ under a continuous counterfactual constraint.

$$\text{minimize } L(g)$$

$$\text{s.t. } \sum_{u \in \mathcal{U}} \sum_{v \in \mathcal{I}(u)} \int_{\mathbf{x}'} \left| P(y|u, x', v) - P(y|u, x, v) \right| \leq \epsilon_1, \ \|\mathbf{x}' - \mathbf{x}\|_2 \leq \epsilon_2$$
$$(13)$$

where $\mathbf{x}$ is the latent vector embedding of user u's real history x in the latent space, $\mathbf{x}'$ is a latent vector selected from the small ϵ_2-neighbourhood of the embedding $\mathbf{x}$, and the integration can be calculated based on Monte Carlo sampling. All other parameters have the same meaning as Eq.(12).

5.5 Model Learning and Optimization

Directly solving the above constrained optimization problem is challenging, which requires maintaining the constraints during the optimization. We formulate the problem as a tractable optimization problem by relaxing the constraints. Technically, we allow some examples to violate the constraints but we penalize these examples in the loss function during optimization.

For counterfactual reasoning in discrete space, we convert the objective in Eq.(12) to the following Lagrange optimization form:

$$\text{minimize } L(g) + \omega L_c$$

$$L_c = \max\left\{0, \sum_{u \in \mathcal{U}} \sum_{v \in \mathcal{I}(u)} \sum_{x' \in C(u,v)} \left| P(y|u, x', v) - P(y|u, x, v) \right| - \epsilon \right\}$$
$$(14)$$

where ω is a parameter controlling the weight of the constraint.

While for reasoning in continuous space, we relax the constraint for optimization similarly. The difference is that the parameter ϵ_2 in Eq.(13) is used to restrict the distance between counterfactual histories and the real history. This can be controlled during the sampling and selection process of the counterfactual histories in the latent embedding space, which we will introduce in the experiments.

We write the relaxed objective as:

$$\text{minimize } L(g) + \omega L_c$$

$$\text{s.t. } \|\mathbf{x}' - \mathbf{x}\|_2 \leq \epsilon_2$$

$$L_c = \max\left\{0, \sum_{u \in \mathcal{U}} \sum_{v \in \mathcal{I}(u)} \int_{\mathbf{x}'} \left| P(y|u, x', v) - P(y|u, x, v) \right| - \epsilon_1 \right\}$$
$$(15)$$

Still, we apply Monte Carlo integration to get the numerical integration value through random samples.

6 EXPERIMENTS

We conduct experiment to explore the CCF framework from different perspectives. In particular, we aim to answer the following research questions:

- **RQ1**: What is the overall performance of the CCF framework, can CCF improve the recommendation performance?
- **RQ2**: How the different heuristic rules influence the recommendation performance?
- **RQ3**: Is it necessary to select counterfactual examples after they are generated?
- **RQ4**: What is the impact of the counterfactual constraint in the learning objective and how strict should the counterfactual constraints be applied?

We will fist describe the datasets, baselines and implementation details and then provide our answers to the above questions.

6.1 Data Description

Our experiments are conducted on two types of datasets. The first type is frequently used benchmark datasets with standard training/validation/testing split to show that our framework is capable of improving the performance in a normal experimental setting. In particular, we use the **MovieLen-100k**[1] and **Amazon Baby**[2] datasets—one is a movie recommendation dataset and the other is an e-commerce dataset. For the second type of datasets, to better show that our framework can help models to capture the real preference of users, we apply our framework on the **Yahoo! R3**[3] and **Coat Shopping**[4] datasets. A special property of these two datasets is that their testing data are collected from randomized trials, i.e., users are recommended with random items and their feedback on these random items are collected to construct the testing dataset.

For all four datasets, following common settings, we consider ratings ≥ 4 as positive feedback (likes) and ratings ≤ 3 as negative feedback (dislikes). For **MovieLens-100k** and **Amazon Baby** datasets, we apply leave-one-out to split the dataset. Specifically, we chronologically sort the interactions for each user and put the latest two positive interactions of each user into the validation set and testing set, respectively, and keep the remaining data in the training set. For **Yahoo! R3** and **Coat Shopping** datasets, since they already have the testing set from randomized trials, we only need to create the training and validation sets. For each user, we randomly select a positive interaction from the history and put into

---

[1] https://grouplens.org/datasets/movielens/
[2] https://nijianmo.github.io/amazon/
[3] https://webscope.sandbox.yahoo.com/catalog.php?datatype=r
[4] https://www.cs.cornell.edu/~schnabts/mnar/

the validation set, all other interactions are put into the training set. The statistics of the datasets are summarized in Table 2.

6.2 Baseline Models

To show the effectiveness of our framework on different models, we apply our framework on six recommendation models, which belong to three different categories, including two matching models (BPR-MF and NCF), two sequential models (GRU4Rec and STAMP), as well as two reasoning models (NLR and NCR).

- **BPR-MF** [32]: The Bayesian Personalized Ranking model for recommendation. We use Matrix Factorization (MF) [20] as the prediction function under the BPR framework, which considers user, item and global bias terms for MF.
- **NCF** [10]: A neural network-based CF method, which adopts a non-linear prediction network for user and item matching.
- **GRU4Rec** [11]: A session-based recommendation model, which uses recurrent neural networks—in particular, Gated Recurrent Units (GRU)—to capture sequential patterns.
- **STAMP** [26]: The Short-Term Attention/Memory Priority model, which uses the attention mechanism to model both short-term and long-term user preferences.
- **NLR** [39]: A reasoning-based model, which adopts Logic-Integrated Neural Network (LINN) to integrate the power of deep learning and logic reasoning for recommendation.
- **NCR** [5]: The Neural Collaborative Reasoning model, which organizes the logic expressions as neural networks for reasoning and prediction in a continuous space.

We also employ two different types of causal frameworks for comparison. In particular, a direct intervention model (CausE) and an Inverse Propensity Scoring-based model (IPS). Both CausE and IPS can be applied on each of the above recommendation models.

- **CausE** [4]: The Causal Embedding framework for recommendation. It splits the collected data into control data and treatment data, and expose as uniformly as possible each user to each item in the treatment data to mimic direct intervention. It jointly learns the representations in the control and treatment data, and finally adopts the control representations for recommendation.
- **IPS** [35]: This is an IPS-based framework that uses propensity scores to re-weight the training samples. It uses a user-independent propensity estimator to estimate the propensity scores for each item and to re-weight each sample.

Finally, we test five versions of our CCF framework (Section 5.2.1 and Section 5.4). Also, each version can be applied on each of the six recommendation models.

- **CCF$_{K1}$**: CCF in discrete space under the Keep One heuristic rule for counterfactual example generation.
- **CCF$_{D1}$**: CCF in discrete space under the Delete One rule.
- **CCF$_{R1r}$**: CCF in discrete space under the Replace One rule. The replacement is conducted randomly.
- **CCF$_{R1n}$**: CCF in discrete space under the Replace One rule. Item is replaced with its nearest neighbour.
- **CCF$_C$**: CCF in continuous space.

Table 2: The Statistics of the datasets

Dataset	# Users	# Items	# Positive	# Negative
ML100k	943	1,682	55,375	44,625
Amazon Baby	19,445	7,050	126,525	34,267
Yahoo! R3	15,400	1,000	129,826	235,878
Coat Shopping	290	300	2,765	8,835

6.3 Implementation Details

We use the same training, validation and testing sets for all recommendation models and frameworks. The models are evaluated on the Top-K Recommendation task with metrics nDCG@10 and Hit@1 (Hit Ratio). We use the pair-wise learning strategy to train all models. In detail, for each positive interaction in training set, we randomly sample an item as the negative sample. For each user in the validation and testing set, we randomly sample 100 negative items for ranking evaluation. Here the negative samples are either negative feedback items (items that user dislikes) or non-interacted items. To fairly evaluate the improvement brought by our CCF framework and the CausE/IPS frameworks, we keep the basic parameters (e.g. learning rate, neural network construction, weight of ℓ_2-regularizer) the same before and after applying the frameworks on the recommendation models. Specifically, we set the embedding dimension as 64, the structure of neural network as two-layer MLP with dimension 64 for all recommendation models (exepct for BPR-MF which is a shallow model). For all baseline recommendation models, we consider the learning rate from {0.0005,0.001,0.003,0.005}, the ℓ_2-regularization weight is chosen from {1e-3, 1e-4, 1e-5}. For sequential models and reasoning models, the history includes previously interacted items even if the item has negative feedback, and the maximum length of history is 10. All parameters are tuned to the best on the validation set based on the measure nDCG@10.

Our framework is applied in both discrete and continuous item space. For discrete version, let (u, x, v, y) be a training example, when selecting counterfactual histories x_i' for this training example (Section 5.2.2), we randomly sample 100 negative items plus the target item v as the candidate item set (101 in total). If item v is still in the top-k list under the counterfactual history x_i' after recommendation algorithm $g(\cdot, \cdot)$, then we will keep the counterfactual history and create a counterfactual example (u, x_i', v, y). It is worth noting that this process is not testing the model based on testing or validation data. The target item is not from the testing or validation set but the target item of a training example. The selection process aims to make sure that if an item is recommended in the real history, then it should be in the top-k list under counterfactual history.

For sequential models and reasoning models, counterfactual outputs can be obtained by simply changing the input of a fixed model. For matching models, however, we need to retrain the recommendation model on counterfactual training data to get the counterfactual outputs. During the retrain process, all users' counterfactual histories are applied on the training set while the validation and testing sets are unchanged. However, if the target items are still in the training data, then all target items will for sure get higher prediction scores after the model is retrained because the loss function of $g(\cdot, \cdot)$ will purposely optimize the scores of these items. As a result, we cannot identify those cases that the target item's recommendation is caused by the counterfactual history. To solve the problem, we

Table 3: Performance on six recommendation models and four datasets. We evaluate on metrics nDCG@10 and Hit@1. The relative improvement on each metric is calculated against the corresponding original performance. In the summary column, we show the number of improved datasets and the average improvement across four datasets (weighted average by the number of testing users of each dataset). Positive improvements are highlighted in bold and the highest improvement is underlined.

Models		ML100k nDCG@10 value	imp	Hit@1 value	imp	Amazon Baby nDCG@10 value	imp	Hit@1 value	imp	Yahoo! R3 nDCG@10 value	imp	Hit@1 value	imp	Coat Shopping nDCG@10 value	imp	Hit@1 value	imp	Summary nDCG@10 num	avg_imp	Hit@1 num	avg_imp
BPR-MF	Original	0.3647	-	0.1490	-	0.2061	-	0.0762	-	0.2252	-	0.1318	-	0.2042	-	0.1561	-	-	-	-	-
	CausE	0.3784	**3.8%**	0.1651	**10.8%**	0.2107	**2.2%**	0.0808	**6.0%**	0.2254	**0.1%**	0.1281	-2.8%	0.2100	**2.8%**	0.1899	**21.7%**	4/4	**1.89%**	3/4	**4.84%**
	IPS	0.3696	**1.3%**	0.1618	**8.6%**	0.2073	**0.6%**	0.0781	**2.5%**	0.2155	-4.3%	0.1252	-5.0%	0.2261	**10.7%**	0.2068	**32.5%**	3/4	-0.18%	3/4	**2.02%**
	CCF_{K1}	0.3661	**0.4%**	0.1554	**4.3%**	0.2039	-1.1%	0.0766	**0.5%**	0.2240	-0.5%	0.1252	-5.0%	0.2324	**13.8%**	0.2363	<u>**51.4%**</u>	2/4	-0.54%	3/4	**0.69%**
	CCF_{D1}	0.3781	**3.7%**	0.1683	**13.0%**	0.2099	**1.8%**	0.0826	<u>**8.4%**</u>	0.2201	-2.3%	0.1236	-6.2%	0.2061	**0.9%**	0.2152	**37.9%**	3/4	**1.06%**	3/4	<u>**6.27%**</u>
	CCF_{R1r}	0.3673	**0.7%**	0.1554	**4.3%**	0.1948	-5.5%	0.0752	-1.3%	0.2222	-1.3%	0.1215	-7.8%	0.2165	**6.0%**	0.2194	**40.6%**	2/4	-3.86%	2/4	-1.36%
	CCF_{R1n}	0.3734	**2.4%**	0.1533	**2.9%**	0.2064	**0.1%**	0.0793	**4.1%**	0.2257	**0.2%**	0.1342	**1.8%**	0.2089	**2.3%**	0.2110	**35.2%**	4/4	**0.36%**	4/4	**4.16%**
	CCF_C	0.3729	**2.2%**	0.1597	**7.2%**	0.2104	**2.1%**	0.0805	**5.6%**	0.2276	<u>**1.1%**</u>	0.1301	-1.3%	0.2150	**5.3%**	0.2068	**32.5%**	4/4	<u>**1.96%**</u>	3/4	**4.82%**
NCF	Original	0.3504	-	0.1458	-	0.1860	-	0.0660	-	0.2161	-	0.1170	-	0.1985	-	0.1519	-	-	-	-	-
	CausE	0.3680	**5.0%**	0.1715	**17.6%**	0.1891	**1.7%**	0.0681	**3.2%**	0.2171	**0.5%**	0.1240	<u>**6.0%**</u>	0.2140	**7.8%**	0.1983	**30.5%**	4/4	**1.84%**	4/4	**5.54%**
	IPS	0.3558	**1.5%**	0.1511	**3.6%**	0.1936	<u>**4.1%**</u>	0.0684	**3.6%**	0.2124	-1.7%	0.1211	**3.5%**	0.2177	**9.7%**	0.2236	**47.2%**	3/4	**2.76%**	4/4	**4.49%**
	CCF_{K1}	0.3564	**1.7%**	0.1436	-1.5%	0.1899	**2.1%**	0.0683	**3.5%**	0.2038	-5.7%	0.1088	-7.0%	0.2263	**14.0%**	0.2068	**36.1%**	3/4	**0.65%**	2/4	**1.52%**
	CCF_{D1}	0.3726	<u>**6.3%**</u>	0.1511	**3.6%**	0.1906	**2.5%**	0.0702	**6.4%**	0.2172	**0.5%**	0.1215	**3.8%**	0.2235	**12.6%**	0.2152	**41.7%**	4/4	**2.59%**	4/4	<u>**6.35%**</u>
	CCF_{R1r}	0.3603	**2.8%**	0.1426	-2.2%	0.1847	-0.7%	0.0665	**0.8%**	0.2183	<u>**1.0%**</u>	0.1182	**1.0%**	0.2027	**2.1%**	0.2068	**36.1%**	3/4	**0.01%**	3/4	**1.33%**
	CCF_{R1n}	0.3717	**6.1%**	0.1522	**4.4%**	0.1879	**1.0%**	0.0671	**1.7%**	0.2171	**0.5%**	0.1186	**1.4%**	0.2046	**3.1%**	0.1688	**11.1%**	4/4	**1.35%**	4/4	**2.05%**
	CCF_C	0.3701	**5.6%**	0.1554	**6.6%**	0.1913	**2.8%**	0.0675	**2.3%**	0.2179	**0.8%**	0.1207	**3.2%**	0.2358	**18.8%**	0.2278	<u>**50.0%**</u>	4/4	**2.93%**	4/4	**3.84%**
GRU4Rec	Original	0.4087	-	0.1865	-	0.2473	-	0.1045	-	0.1682	-	0.0907	-	0.1147	-	0.0759	-	-	-	-	-
	CausE	0.4111	**0.6%**	0.1908	**2.3%**	0.2446	-1.1%	0.1021	-2.3%	0.1747	**3.9%**	0.0952	<u>**5.0%**</u>	0.1157	**0.9%**	0.0802	**5.7%**	3/4	**0.15%**	3/4	-0.20%
	IPS	0.4136	**1.2%**	0.1876	**0.6%**	0.2501	**1.1%**	0.1060	**1.4%**	0.1788	**6.3%**	0.0911	**0.4%**	0.1160	**1.1%**	0.0802	**5.7%**	4/4	**2.22%**	4/4	**1.21%**
	CCF_{K1}	0.4225	**3.4%**	0.2015	**8.0%**	0.2605	**5.3%**	0.1165	**11.5%**	0.1714	**1.9%**	0.0915	**0.9%**	0.1170	**2.0%**	0.0675	-11.1%	4/4	**4.35%**	3/4	**8.48%**
	CCF_{D1}	0.4281	<u>**4.7%**</u>	0.1972	**5.7%**	0.2555	**3.3%**	0.1069	**2.3%**	0.1707	**1.5%**	0.0850	-6.3%	0.1226	**6.9%**	0.0886	**16.7%**	4/4	**3.10%**	4/4	**1.04%**
	CCF_{R1r}	0.4241	**3.8%**	0.1972	**5.7%**	0.2752	**11.3%**	0.1237	**18.4%**	0.1820	**8.2%**	0.0924	**1.9%**	0.1299	**13.3%**	0.0802	**5.7%**	4/4	**10.07%**	4/4	**13.57%**
	CCF_{R1n}	0.4235	**3.6%**	0.2015	**8.0%**	0.2519	**1.9%**	0.1073	**2.7%**	0.1766	**3.4%**	0.0940	**5.0%**	0.1264	**10.2%**	0.0802	**5.7%**	4/4	**2.53%**	4/4	**3.69%**
	CCF_C	0.4238	**3.7%**	0.2015	**8.0%**	0.2528	**2.2%**	0.1070	**2.4%**	0.1743	**3.6%**	0.0936	**3.2%**	0.1352	<u>**17.9%**</u>	0.0970	**27.8%**	4/4	**2.95%**	4/4	**3.56%**
STAMP	Original	0.4016	-	0.1758	-	0.2418	-	0.1042	-	0.1766	-	0.0899	-	0.1182	-	0.0675	-	-	-	-	-
	CausE	0.4202	**4.6%**	0.2079	**18.3%**	0.2422	**0.2%**	0.1069	**2.6%**	0.1700	-3.7%	0.0936	**4.1%**	0.1235	**4.5%**	0.0844	**25.5%**	3/4	-0.18%	4/4	**4.68%**
	IPS	0.4094	**1.9%**	0.1940	**4.0%**	0.2359	-2.4%	0.0970	-6.9%	0.1726	-2.3%	0.0846	-5.9%	0.1251	**5.8%**	0.0928	**37.5%**	2/4	-1.86%	2/4	-4.87%
	CCF_{K1}	0.3849	-4.2%	0.1586	-9.8%	0.2641	**9.2%**	0.1124	**7.9%**	0.1736	-1.7%	0.0932	**3.7%**	0.1222	**3.4%**	0.0633	-6.2%	3/4	**6.06%**	2/4	**5.26%**
	CCF_{D1}	0.4103	**2.2%**	0.1876	**6.7%**	0.2710	**12.1%**	0.1191	**14.3%**	0.1736	-1.7%	0.0977	**8.7%**	0.1236	**4.6%**	0.0844	**25.0%**	3/4	**8.18%**	4/4	**12.70%**
	CCF_{R1r}	0.4258	**6.0%**	0.2047	**16.4%**	0.2723	**12.6%**	0.1213	**16.4%**	0.1809	**2.4%**	0.0989	<u>**10.0%**</u>	0.1261	**6.7%**	0.0717	**6.2%**	4/4	**9.76%**	4/4	<u>**14.82%**</u>
	CCF_{R1n}	0.4203	**4.7%**	0.1972	**12.2%**	0.2678	**10.8%**	0.1177	**13.0%**	0.1807	**2.3%**	0.0956	**6.3%**	0.1243	**5.2%**	0.0970	<u>**43.7%**</u>	4/4	**8.37%**	4/4	**12.14%**
	CCF_C	0.4113	**2.4%**	0.1876	**6.7%**	0.2674	**10.6%**	0.1169	**12.2%**	0.1776	**0.6%**	0.0920	**2.3%**	0.1306	<u>**10.5%**</u>	0.0802	**18.8%**	4/4	**7.79%**	4/4	**9.77%**
NLR	Original	0.4029	-	0.1886	-	0.2481	-	0.1045	-	0.1660	-	0.0784	-	0.1340	-	0.0802	-	-	-	-	-
	CausE	0.4147	**2.9%**	0.1940	**2.9%**	0.2646	**6.7%**	0.1161	**11.1%**	0.1703	**2.6%**	0.0932	**18.9%**	0.1253	-6.5%	0.0886	**10.5%**	3/4	**5.24%**	4/4	**12.08%**
	IPS	0.4069	**1.0%**	0.1940	**2.9%**	0.2579	**4.0%**	0.1106	**5.8%**	0.1712	**3.1%**	0.0858	**9.4%**	0.1327	-1.0%	0.1097	**36.8%**	3/4	**3.46%**	4/4	**6.98%**
	CCF_{K1}	0.4064	**0.9%**	0.1929	**2.3%**	0.2668	**7.5%**	0.1125	**7.7%**	0.1667	**0.4%**	0.0862	**9.9%**	0.1377	**2.8%**	0.0759	-5.4%	4/4	**5.34%**	3/4	**7.46%**
	CCF_{D1}	0.4049	**0.5%**	0.1919	**1.7%**	0.2735	<u>**10.2%**</u>	0.1213	**16.1%**	0.1690	**1.8%**	0.0829	**5.7%**	0.1241	-7.4%	0.0928	**15.7%**	3/4	<u>**7.25%**</u>	4/4	**12.69%**
	CCF_{R1r}	0.4083	**1.3%**	0.1951	**3.4%**	0.2636	**6.2%**	0.1155	**10.5%**	0.1702	**2.5%**	0.0821	**4.7%**	0.1375	**2.6%**	0.0717	-10.6%	4/4	**4.93%**	3/4	**8.24%**
	CCF_{R1n}	0.4042	**0.3%**	0.1822	-3.4%	0.2691	**8.5%**	0.1224	<u>**17.1%**</u>	0.1742	**4.9%**	0.0813	**3.7%**	0.1256	-6.3%	0.0928	**15.7%**	4/4	**6.75%**	4/4	**12.53%**
	CCF_C	0.4153	**3.1%**	0.1951	**3.4%**	0.2692	**8.5%**	0.1214	**16.2%**	0.1709	**3.0%**	0.0866	**10.5%**	0.1369	**2.2%**	0.0844	**5.2%**	4/4	**6.75%**	4/4	<u>**13.71%**</u>
NCR	Original	0.4227	-	0.1972	-	0.3192	-	0.2687	-	0.2673	-	0.1063	-	0.2608	-	0.0506	-	-	-	-	-
	CausE	0.4234	**0.2%**	0.2090	**6.0%**	0.4631	**45.1%**	0.4561	**69.7%**	0.2630	-1.6%	0.1030	-3.1%	0.2813	**7.9%**	0.0886	**75.1%**	3/4	**30.67%**	3/4	**49.04%**
	IPS	0.4237	**0.2%**	0.2036	**3.2%**	0.3983	**24.8%**	0.2633	-2.0%	0.2681	**0.3%**	0.1154	**8.6%**	0.2916	**11.8%**	0.1350	<u>**166.8%**</u>	4/4	**17.28%**	3/4	**4.20%**
	CCF_{K1}	0.4094	-3.1%	0.1940	-1.6%	0.4416	**38.3%**	0.4335	**61.3%**	0.2709	**1.3%**	0.1100	**3.5%**	0.2896	**11.0%**	0.1139	**125.1%**	3/4	**26.44%**	3/4	**45.13%**
	CCF_{D1}	0.4144	-2.0%	0.1897	-3.8%	0.4613	**44.5%**	0.4360	**62.3%**	0.2679	**0.2%**	0.1208	<u>**14.5%**</u>	0.3098	**18.8%**	0.1308	**158.5%**	3/4	**30.69%**	4/4	**48.67%**
	CCF_{R1r}	0.4271	**1.0%**	0.2004	**1.6%**	0.4415	**38.3%**	0.4124	**53.5%**	0.2705	**1.2%**	0.1183	**11.3%**	0.2874	**10.2%**	0.1013	**100.2%**	4/4	**26.73%**	4/4	**41.20%**
	CCF_{R1n}	0.4195	-0.8%	0.2111	<u>**7.0%**</u>	0.4290	**34.4%**	0.3961	**47.4%**	0.2677	**0.1%**	0.1175	**10.5%**	0.2816	**8.0%**	0.0970	**91.7%**	3/4	**23.64%**	4/4	**37.13%**
	CCF_C	0.4274	<u>**1.1%**</u>	0.2058	**4.4%**	0.4683	<u>**46.7%**</u>	0.4572	<u>**70.2%**</u>	0.2718	<u>**1.7%**</u>	0.1174	**10.4%**	0.3095	**18.7%**	0.1266	**150.2%**	4/4	<u>**32.77%**</u>	4/4	<u>**53.70%**</u>

select the target items that only appear in the original training data but not in the counterfactual training data. More specifically, the target items would be the deleted (for D1 and K1 rules) or replaced (for R1 rule) items. Those items will get higher scores in the original model because they appear in the original training set but their scores in the retrained model are not guaranteed because they do not appear in the counterfactual training set. The counterfactual history selection process is controlled by parameter k (top-k list). The value of k is tuned in {50,60,70,80,90,100} based on validation set, and we will study the influence of k in the experiments.

For continuous version, the counterfactual examples are generated in the latent embedding space. We sample 10 counterfactual embeddings in the ϵ_2-neighbourhood to estimate the integrate in Eq.(15). More specifically, to sample each counterfactual embedding, we first sample a noise vector θ from a Gaussian distribution, and randomly pick a number γ that satisfies $0 \leq \gamma \leq \epsilon_2$. To make sure the constraint $\|\mathbf{x}' - \mathbf{x}\|_2 \leq \epsilon_2$ is satisfied, we define $\mathbf{x}'$ as $\mathbf{x} + \sqrt{\gamma}\theta/\|\theta\|_2$. The counterfactual constraint weight ω is tuned in {0.001,0.1,0.5,1.0}, the counterfactual constraint threshold ϵ (ϵ_1 for continuous version) is in {0.1,0.5,1.0}. For continuous version, the

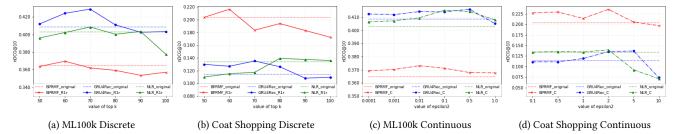| (a) ML100k Discrete | (b) Coat Shopping Discrete | (c) ML100k Continuous | (d) Coat Shopping Continuous |

Figure 3: nDCG@10 for ML100k and Coat Shopping datasets with different counterfactual selection parameters. (a) and (b) are discrete versions with R1r heuristic rule under parameter k. (c) and (d) are continuous versions under parameter ϵ_2.

parameter ϵ_2 in Eq.(15) is selected in $\{0.1, 1.0, 2.0, 5.0\}$. We will show the influence of the parameters in the following experiments.

6.4 Overall Performance

The overall performance of applying causal frameworks (CausE, IPS, CCF$_*$) on six recommendation models over four datasets are shown in Table 3, including the ranking metrics (nDCG and Hit Ratio) and relative improvements against the original performance of the recommendation model. All positive improvements are shown in bold numbers, the best improvement for each recommendation model is underlined. The last column of the table shows the summary over four datasets, including on how many of the four datasets a causal framework improved the performance, and the average improvement over the four datasets. Average calculation is weighted by the number of testing users of each dataset, and negative improvements are also counted in when calculating the averages.

From the results we can see that in most cases causal frameworks can bring positive improvement to the recommendation models. In particular, for most recommendation models, the CCF framework brings improvements on 3 or 4 out of the four datasets. Comparing CausE, IPS and CCF, we see that for all of the six recommendation models, the largest average improvement over four datasets is always brought by our CCF framework (see the underlined numbers in the last column). For matching models (BPR-MF and NCF), the average improvement brought by CCF is about 2% on nDCG@10 and 6% on Hit@1; for sequential models (GRU4Rec and STAMP), the average improvement brought by CCF is about 10% on nDCG@10 and 14% on Hit@1; for reasoning models (NLR and NCR), the average improvement can be as large as 32% on nDCG@10 and 53% on Hit@1. As we mentioned before, CausE improves performance by splitting the observational training data into approximately randomized data and IPS improves performance by re-weighting the observational training data. Both of the two frameworks only consider the real-world examples, however, our CCF framework not only considers real-world examples but also involves counterfactual examples, which helps to better capture the user preference and improve the recommendation performance.

Considering the two types of datasets, *ML100k* and *Amazon Baby* are traditional training-validation-testing split datasets, while *Yahoo! R3* and *Coat Shopping* are datasets whose testing sets are collected from randomized trials. One minor but noteworthy observation is that matching models (BPR-MF and NCF) get better performance than sequential models (GRU4Rec and STAMP) on *Yahoo! R3* and *Coat Shopping*. The reason is that the training sets

of the two datasets do not have timestamp information, and thus the user history interactions are randomly ordered. For sequential models, this means that the model cannot leverage the item ordering information and thus results in sacrificed performance, which is consistent with the observations in [51]. On *ML100k* and *Amazon Baby*, where the sequential models can be properly executed, we can see that sequential models are better than matching models. However, reasoning models such as the NCR model is better than both matching and sequential models on both two types of datasets. This is because the reasoning model randomizes the events in the premise during model training and thus does not rely on timestamp or item ordering information.

In this work, we care more about the performance improvement against each model itself when applying a causal framework on top of the model. Considering different versions of the CCF framework, we see that all versions can improve the performance in most cases. Therefore, performance improvement would not be a major concern when deciding which version to apply. Actually, even though our experiments have considered four discrete versions and one continuous version, there may still exist other ways to apply counterfactual reasoning in discrete or continuous space, as long as the method of generating counterfactual examples are well defined. This shows the flexibility of our counterfactual reasoning framework. In the next subsection, we will compare the different CCF versions in detail.

6.5 Analyzing Counterfactual Examples

Generating and selecting counterfactual examples are essential components of the CCF framework (Section 5.2). In this section, we aim to answer the research questions **RQ2** and **RQ3**. Specifically, we will discuss the influence of the counterfactual examples in two perspectives. We will first dig into the difference between different heuristic rules for generating counterfactual examples in the discrete space. We then focus on selecting counterfactual examples to show the necessity of the selection process after generation.

*6.5.1 **Difference between Heuristic Rules**.* According to the performance shown in Table 3, the CCF framework has the ability to improve the performance of a base recommendation model. How to generate counterfactual examples plays an important role here. In this section, we focus on the discrete versions of our framework and discuss the effect of different heuristic rules.

We have three types of heuristic rules in Table 1, including the Keep One (K1), Delete One (D1) and Replace One (R1). Besides, the
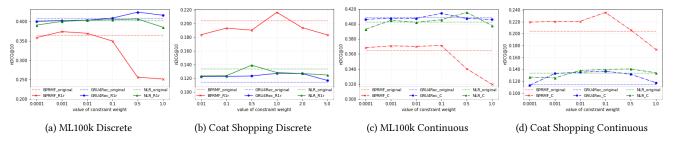
| (a) ML100k Discrete | (b) Coat Shopping Discrete | (c) ML100k Continuous | (d) Coat Shopping Continuous |

Figure 4: nDCG@10 for ML100k and Coat Shopping datasets with different counterfactual constraint weight ω. (a) and (b) are discrete versions with R1r heuristic rule. (c) and (d) are continuous versions.

R1 rule has two variations—R1-random and R1-nearest. Among the heuristic rules, the K1 and D1 rules generate much fewer number of counterfactual histories than R1, because K1 and D1 are limited by the number of interactions in the user's real history, while the R1 rule can replace each interacted item with a great number of possible items in the space, thus creating many counterfactual histories. As a result, it is more difficult for K1 and D1 to get qualified counterfactual examples in the selection process when k is small, where k is the top-k selection threshold introduced in Section 5.2.2. Considering the difficulty of generating qualified counterfactual examples, the R1 rules are intuitively better than D1 and K1.

This is consistent with the experimental results. Crossing all models and datasets in Table 3, we see that the R1 rule is more likely to get better performance compared with other rules. Summarizing the numbers in Table 3, among the 2 (measures) × 4 (datasets) × 6 (models) = 48 cases, K1 achieves 35 positive improvement cases out of 48, D1 achieves 41 out of 48, R1-nearest achieves 45 out of 48, and R1-random achieves 41 out of 48. Overall, all heuristic rules have the potential to improve a base recommendation algorithm, and the key to choose a heuristic rule is what practical meaning of the counterfactual examples is required in the system.

Among the heuristic rules, R1 and D1 only change one item in the history, while K1 changes many items since only one item is kept. As a result, D1 and R1 apply minimal changes to the history in terms of the number of items, while K1 applies a relatively more significant change. However, according to Table 3, K1 is still able to improve the performance in most cases. This benefit is brought by the selection process after the counterfactual examples are generated. We will discuss about it in the next subsection.

6.5.2 Counterfactual Example Selection. For the CCF discrete versions, the selection is based on the parameter k, i.e., the counterfactual example will be selected if the target item is ranked to top-k under the counterfactual history (Section 5.2.2). For the CCF continuous version, the selection is based on the parameter ϵ_2 in Eq.(15), i.e., a counterfactual history embedding is selected only if it is close enough to the real history embedding.

We first examine the discrete versions. We plot the nDCG@10 under different k (from 50 to 100) and under the heuristic rule R1r in Figure 3(a-b) for two types of datasets. Other rules and datasets have similar observations. Since we randomly sample 100 negative items for ranking during selection, so when k is set as 100, we are actually doing no selection because the target item is almost surely in top-100 among the 101 items. We can see that when k is properly

chosen, the selected counterfactual examples will improve the performance through the counterfactual constraint. However, when k is too large—such as $k = 100$ that all the generated counterfactual examples are selected—the counterfactual constraint will hurt the performance. This observation is consistent with the theory of conditional intervention (Section 5.1). When doing conditional intervention (Eq.(4)(5)), we need to use those counterfactual histories x' that lead to the target item v still being recommended for user u under the recommendation algorithm $v = g(u, x')$, and only these counterfactual histories are required to reach similar predictions with the real history in the counterfactual constraint (Eq.(12)(14)). If too many "irrelevant" counterfactual histories are forced to make the same predictions, it will violate the assumptions of conditional intervention and thus lead to decreased performance.

For continuous version, we plot the nDCG@10 under different ϵ_2 in Figure 3(c-d). When ϵ_2 is very small, the generated counterfactual embedding $\mathbf{x}'$ is very close to the real history embedding $\mathbf{x}$ (Eq.(13)(15)), therefore, the performance after applying the CCF framework has no much difference from the original recommendation performance, since the counterfactual constraint in Eq.(13)(15) is easily satisfied. In contrast, if ϵ_2 is too large, the generated counterfactual embeddings are too far away from the real embedding, and when we force their predictions to be close, the performance will decrease. The reason is similar to the discrete versions. As a result, for both discrete and continuous versions, appropriate selection of the counterfactual examples is important, but the selection can be conducted in either discrete or continuous space.

As mentioned in the previous subsection, the K1 rule applies a relatively more significant change to the user history when generating counterfactual histories, but CCF$_{K1}$ is still able to improve the performance in many cases (35 out of 48 cases in Table 3). This also benefits from the the selection process. This process will make sure that the selected counterfactual histories do not make significant changes in terms of the recommendation output. In other words, although K1 applies a significant change in terms of the number of history items, the selected counterfactual examples are still similar to the real example in terms of generating similar recommendations, and thus to satisfy the requirement of *conditional* intervention.

Overall, counterfactual examples are fundamental elements of the CCF framework. The counterfactual examples go through two steps: step one generates candidate counterfactual examples and step two selects the qualified examples to satisfy the requirements of conditional intervention. The generation and selection process can be conducted in both discrete and continuous space.
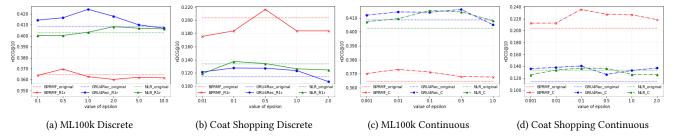
Figure 5: nDCG@10 for ML100k and Coat Shopping datasets with different counterfactual constraint threshold ϵ (ϵ_1 for continuous version). (a) and (b) are discrete versions with R1r heuristic rule. (c) and (d) are continuous versions.

6.6 Analyzing Counterfactual Constraints

After the counterfactual examples are selected, we apply them into the counterfactual constraint for counterfactual learning. There are two important parameters for the constraint, one is parameter ω in Eq.(14) and (15) which controls the importance of the constraint in the learning objective, and the other is the parameter ϵ in Eq.(14) (or ϵ_1 in Eq.(15)) which controls how strict the counterfactual constraints are. In this section, we provide the answers to **RQ4**. We will discuss the two parameters separately in the following.

*6.6.1 **Counterfactual Constraint Weight**.* Our framework aims to learn the loss function of a base recommendation algorithm under a counterfactual constraint (Eq.(12)(13)), but for optimization, we relax the constraint and convert the loss function into Eq.(14) and (15). The larger the counterfactual constraint weight ω, the more likely the results will follow the constraint.

We tune the constraint weight ω while keeping other parameters fixed. The results of nDCG@10 are shown in Figure 4 for both discrete and continuous versions on two types of datasets. We see that in most cases the performance would first lift and then drop with the increase of the ω. This result shows that the counterfactual constraint is useful for improving performance, but it also requires a good balance with the recommendation loss. When the weight is too small, the counterfactual constraint has little effect on the total loss, leading to only slight improvement or even slight loss considering the larger model complexity. In contrast, when the weight is too large, the constraint loss will dominate the total loss, and thus the recommendation performance is significantly decreased since the recommendation loss does not take too much effect. Compared with an overly large weight, a smaller weight than the proper value would have no or less hurt on the performance. In summary, counterfactual constraint will help improve the recommendation performance, but the weight needs to be carefully specified.

*6.6.2 **Counterfactual Constraint Threshold**.* The counterfactual constraint threshold (i.e. ϵ in Eq.(12) and ϵ_1 in Eq.(13)) controls how rigorous is the constraint. After converting the counterfactual constraint into the optimization loss (Eq.(14)(15)), the constraint loss will not be optimized when the difference is already less than the threshold. Smaller threshold is supposed to result in closer prediction scores between counterfactual examples and the real example, while larger threshold will lead to a more relaxed constraint.

We plot the nDCG@10 with different threshold in Figure 5. We see that the performance would first increase and then decrease and finally tend to be flat when the threshold is large enough. When

the threshold is too small, the constraint would be too tight and it only allows for small variations between counterfactual and real examples' predictions. This makes the model less capable of handling the potential errors in counterfactual examples and thus leads to only slightly improved or even slightly decreased performance. When the threshold is too large, we are actually applying no counterfactual constraint, since the difference would always be smaller than the threshold and thus the L_c in Eq.(14) and (15) would be 0 in most cases. As a result, the performance becomes relatively flat when the threshold is large enough.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a Causal Collaborative Filtering (CCF) framework for personalized recommendation. We first show the $P(y|u, do(v))$ formulation of CCF and further show that many traditional collaborative filtering algorithms are actually special cases of CCF under simplified causal graphs. We then provided a conditional intervention approach to estimating $P(y|u, do(v))$ based on observational data, and further proposed a counterfactual constrained learning framework to make counterfactual reasoning possible under a standard machine learning pipeline. Experimental results on both split and randomized trail datasets show that CCF can improve the performance of the matching-, sequential- and reasoning-based recommendation models in most cases and by large margins.

The CCF framework is a very flexible framework and it can be extended in various dimensions. First, we can further extend the causal graph in Figure 1(d) into more complicated causal graphs, so that we can estimate $P(y|u, do(v))$ for more complex recommendation scenarios. Second, our proposed conditional intervention and counterfactual learning approach may also be applied to other intelligent tasks such as vision and language learning. Third, except for the conditional intervention and counterfactual learning approach, there may exist other approaches to estimating $P(y|u, do(v))$ based on observational data, which we will explore in the future. Finally, we only used the user interaction information in this work, while in the future, we can consider the rich multimodal information in recommender systems for causal modeling, such as the user reviews, images and item descriptions, and this may also help us to design explainable recommendation models based on causal learning.

in this material are those of the authors and do not necessarily reflect those of the sponsors.

REFERENCES

[1] Himan Abdollahpouri, Robin Burke, and Bamshad Mobasher. 2017. Controlling popularity bias in learning-to-rank recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*. 42–46.

[2] Qingyao Ai, Vahid Azizi, Xu Chen, and Yongfeng Zhang. 2018. Learning heterogeneous knowledge base embeddings for explainable recommendation. *Algorithms* 11, 9 (2018), 137.

[3] Jonathan Baxter. 2000. A model of inductive bias learning. *Journal of artificial intelligence research* 12 (2000), 149–198.

[4] Stephen Bonner and Flavian Vasile. 2018. Causal embeddings for recommendation. In *Proceedings of the 12th ACM Conference on Recommender Systems*. 104–112.

[5] Hanxiong Chen, Shaoyun Shi, Yunqi Li, and Yongfeng Zhang. 2021. Neural Collaborative Reasoning. In *Proceedings of the 30th Web Conference (WWW)*.

[6] Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiaxi Tang, Yixin Cao, Zheng Qin, and Hongyuan Zha. 2018. Sequential recommendation with user memory networks. In *Proceedings of the eleventh ACM international conference on web search and data mining*. 108–116.

[7] 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 *Proceedings of the 1st workshop on deep learning for recommender systems*. 7–10.

[8] Michael D Ekstrand, John T Riedl, and Joseph A Konstan. 2011. *Collaborative filtering recommender systems*. Now Publishers Inc.

[9] Azin Ghazimatin, Oana Balalau, Rishiraj Saha Roy, and Gerhard Weikum. 2020. PRINCE: Provider-side Interpretability with Counterfactual Explanations in Recommender Systems. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 196–204.

[10] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*. 173–182.

[11] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based recommendations with recurrent neural networks. In *International Conference on Learning Representations*.

[12] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based recommendations with recurrent neural networks. *ICLR* (2016).

[13] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge Belongie, and Deborah Estrin. 2017. Collaborative metric learning. In *In WWW*. 193–201.

[14] Yitong Ji, Aixin Sun, Jie Zhang, and Chenliang Li. 2020. A Re-visit of the Popularity Baseline in Recommender Systems. *SIGIR* (2020).

[15] Thorsten Joachims, Adith Swaminathan, and Maarten de Rijke. 2018. Deep learning with logged bandit feedback. In *International Conference on Learning Representations*.

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

[17] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE.

[18] Alexandros Karatzoglou, Xavier Amatriain, Linas Baltrunas, and Nuria Oliver. 2010. Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*. 79–86.

[19] Joseph A Konstan, Bradley N Miller, David Maltz, Jonathan L Herlocker, Lee R Gordon, and John Riedl. 1997. GroupLens: applying collaborative filtering to Usenet news. *Commun. ACM* 40, 3 (1997), 77–87.

[20] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.

[21] Matt J Kusner, Joshua Loftus, Chris Russell, and Ricardo Silva. 2017. Counterfactual fairness. In *Advances in Neural Information Processing Systems*. 4066–4076.

[22] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. 2017. Neural attentive session-based recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM, 1419–1428.

[23] Dawen Liang, Laurent Charlin, and David M Blei. 2016. Causal inference for recommendation. In *Causation: Foundation to Application, Workshop at UAI*. AUAI.

[24] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing* 7, 1 (2003).

[25] Dugang Liu, Pengxiang Cheng, Zhenhua Dong, Xiuqiang He, Weike Pan, and Zhong Ming. 2020. A general knowledge distillation framework for counterfactual recommendation via uniform data. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 831–840.

[26] Qiao Liu, Yifu Zeng, Refuoe Mokhosi, and Haibin Zhang. 2018. STAMP: short-term attention/memory priority model for session-based recommendation. In *KDD*. 1831–1839.

[27] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. 2015. Image-based recommendations on styles and substitutes. In *SIGIR*. ACM.

[28] Andriy Mnih and Russ R Salakhutdinov. 2008. Probabilistic matrix factorization. In *Advances in neural information processing systems*. 1257–1264.

[29] Judea Pearl. 2000. Causality: Models, reasoning and inference. *Cambridge University Press* (2000).

[30] Judea Pearl, Madelyn Glymour, and Nicholas P Jewell. 2016. *Causal inference in statistics: A primer*. John Wiley & Sons.

[31] Steffen Rendle. 2010. Factorization machines. In *2010 IEEE International Conference on Data Mining*. IEEE, 995–1000.

[32] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2012. BPR: Bayesian personalized ranking from implicit feedback. *UAI* (2012).

[33] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*. 811–820.

[34] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. 1994. GroupLens: an open architecture for collaborative filtering of netnews. In *CSCW*. 175–186.

[35] Yuta Saito, Suguru Yaginuma, Yuta Nishino, Hayato Sakata, and Kazuhide Nakata. 2020. Unbiased Recommender Learning from Missing-Not-At-Random Implicit Feedback. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 501–509.

[36] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *WWW*. 285–295.

[37] J Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. 2007. Collaborative filtering recommender systems. In *The adaptive web*. Springer, 291–324.

[38] Tobias Schnabel, Adith Swaminathan, Ashudeep Singh, Navin Chandak, and Thorsten Joachims. 2016. Recommendations as treatments: Debiasing learning and evaluation. In *ICML*.

[39] Shaoyun Shi, Hanxiong Chen, Weizhi Ma, Jiaxin Mao, Min Zhang, and Yongfeng Zhang. 2020. Neural Logic Reasoning. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1365–1374.

[40] Jiaxi Tang and Ke Wang. 2018. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. 565–573.

[41] Pengfei Wang, Hanxiong Chen, Yadong Zhu, Huawei Shen, and Yongfeng Zhang. 2019. Unified Collaborative Filtering over Graph Embeddings. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 155–164.

[42] 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.

[43] Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. 2017. Deep Matrix Factorization Models for Recommender Systems.. In *IJCAI*, Vol. 17. Melbourne, Australia, 3203–3209.

[44] 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 *Proceedings of the 12th ACM Conference on Recommender Systems*. 279–287.

[45] 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 *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 974–983.

[46] Bowen Yuan, Jui-Yang Hsia, Meng-Yuan Yang, Hong Zhu, Chih-Yao Chang, Zhenhua Dong, and Chih-Jen Lin. 2019. Improving ad click prediction by considering non-displayed events. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 329–338.

[47] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative Knowledge Base Embedding for Recommender Systems. In *KDD*.

[48] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative knowledge base embedding for recommender systems. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 353–362.

[49] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)* 52, 1 (2019), 1–38.

[50] Yongfeng Zhang, Qingyao Ai, Xu Chen, and W Bruce Croft. 2017. Joint representation learning for top-n recommendation with heterogeneous information sources. In *CIKM*. 1449–1458.

[51] Wayne Xin Zhao, Junhua Chen, Pengfei Wang, Qi Gu, and Ji-Rong Wen. 2020. Revisiting Alternative Experimental Settings for Evaluating Top-N Item Recommendation Algorithms. In *CIKM*. 2329–2332.

[52] Lei Zheng, Vahid Noroozi, and Philip S. Yu. 2017. Joint deep modeling of users and items using reviews for recommendation. In *WSDM*.