Collaborative Filtering With Representation Learning in the Frequency Domain

Ali Shirali[a,b], Reza Kazemi[c], Arash Amini[b]

[a]*Department of Electrical Engineering and Computer Science, UC Berkeley, CA, USA*
[b]*Department of Electrical Engineering, Sharif University of Technology, Tehran, Iran*
[c]*Electronics Research Institute, Sharif University of Technology, Tehran, Iran*

This work was submitted in Apr. 2023, revised in Sep. 2023 and Feb. 2024, and accepted in Jul. 2024.

Abstract

In the context of recommender systems, collaborative filtering is the method of predicting the ratings of a set of items given by a set of users based on partial knowledge of the ratings. Commonly, items and users are represented via vectors, and to predict ratings, approaches such as vector inner-product (aka matrix factorization) or more advanced nonlinear functions are applied. In this paper, while we adopt the common vectorial representation, we consider a general model in which the ratings are smooth functions of the item representations. Smoothness ensures similar items with nearby vectors will also get similar ratings as we expect from a human rater. We represent user smooth scoring functions in a so-called frequency domain and learn their representations alongside item representations using 1) an iterative optimization approach that maps items and users alternatively, and 2) a feed-forward neural network consisting of interpretable layers. We also address the challenge of the distribution shift from observed to unobserved ratings (aka missing-not-at-random) with insights from the frequency domain. We evaluate the predictive power of our method and its robustness in missed-not-at-random settings on four popular benchmarks. Despite its simplicity and interpretability, our method yields a remarkable performance compared to the state-of-the-art[1].

Keywords: Collaborative filtering, frequency domain, missing not at random, recommender systems, representation learning

[1]All codes are publicly available at: `https://github.com/alishiraliGit/collaborative-filtering-in-the-frequency-domain`

1. Introduction

Nowadays, recommender systems (RS) are among the most effective tools used by large-scale companies for maintaining their current and attracting new customers. It is interesting to mention that 80 percent of watched movies on Netflix [9] and 60 percent of video clicks on Youtube [4] are linked with recommendations. However, the world of RS is well extended beyond the video industry.

Recommender systems are generally categorized into three groups [40]: content-based RS, collaborative filtering (CF), and hybrid RS. In content-based methods, recommendations are made based on the similarity of items or users. Here, the similarity of two items or two users is measured by their *content*, i.e. any static information or prior knowledge about users and items. On the other side, CF, which is the focus of this paper, relies on historical interactions to generate recommendations. In hybrid RS, besides the historical data, the RS has access to some side information from users or items [25]. Cross domain collaborate filtering (CDCF) as a hybrid method is also recommended to alleviate the sparsity problem in the recommender systems [37, 38, 39]. CDCF solves the sparsity problem by transferring rating knowledge from auxiliary domains. In this work, we consider the general setting of the CF in which no side information is available.

CF methods themselves consist of two well-established categories: latent feature models and graph feature models [7]. Latent feature models in general learn an explicit latent vector per each user and item, and predict the rating solely as a function of these latent features. For example, matrix factorization (MF) is a well-known and widely applied latent feature method that predicts the ratings as the inner-product of user and item features. Graph feature models, however, predict the ratings of a set of items based on their underlying similarity graph. This graph might be the similarity graph of items [13, 17] or the bipartite user-item graph [42, 33]. Self-supervised methods are also proposed to augment graph feature models [35, 36].

Due to their interpretability and scalability, latent feature methods are promising approaches. However, these benefits usually come at the expense of neglecting local graph structure of the data. Although most recent advances in CF methods come from improvements in graph feature models, it is still important to design better latent feature models; indeed, many of the advanced CF methods with impressive performance are obtained by incorporating graph features into latent feature models and the true potential of latent feature models is not yet reached [7, 11, 3]. For example, the local low-rank matrix approximation method (LLORMA) [15] relaxes the low-rank assumption of MF using graph features. Generally, we expect improvements in latent feature methods by incorporating them in graph feature augmented methods. Moreover, recent studies indicate that through extensive fine-tuning of latent feature models within the matrix factorization family, nearly every other method can be challenged, underscoring the capability of these models [20].

In this work, we present a new frequency-domain viewpoint on the latent feature approach. More precisely, we assign each user a scoring *function* that takes an item (or its representation) as the input and outputs the rating of the user for that particular item. We first show many of the well-known latent feature methods fit into this model as special cases for scoring functions that are not necessarily optimal. Next, we look into the CF problem as interpolating/extrapolating these unknown scoring functions based on a few samples. As the sample consistency is not even nearly enough for interpolating/extrapolating the scoring functions, we restrict the space of feasible functions to be *smooth* as well as bandlimited around the zero-frequency (the bandwidth shall be determined). We believe this is a natural choice that makes the recovery problem feasible and motivate it in Section 2.

Estimating (recovering) the scoring functions is a non-convex optimization problem. In Section 3, we propose two techniques to effectively solve this problem. While the first approach uses conventional alternating approaches for the minimization (Section 3.1), the second approach makes use of deep neural networks (Section 3.2).

Recently, deep learning has found its way to RS, and specifically CF methods [40]. Deep networks are able to learn non-linear representations with powerful optimization tools, and their efficient implementations have made them promising CF approaches. We propose the *SmoothRecNet* architecture which is a feed-forward network and has the advantage of interpretability. Here, interpretability implies that a system operator could predict the effect of various manipulations on the output in advance [41].

A well-documented challenge in data-driven RS systems is the non-uniform spread of the available observed scores; this challenge is commonly referred to as the missing-not-at-random (MNAR) problem. Indeed, users frequently tend to report feedback in positive cases or considerably negative cases. Another cause is that the data itself is collected by another recommender system which is automatically biased towards favorable items [19, 16]. It is shown that in this case, samples in the dataset are no longer independent and this can affect the evaluation of newly proposed recommenders [26].

There are two popular ways of addressing the MNAR problem. A direct way is to treat the recommendation problem as missing-data imputation of the rating matrix based on the joint likelihood of the missing data and the ratings [12]. This usually leads to sophisticated methods. Alternatively, inverse propensity scoring (IPS) has been shown to effectively debias the training and evaluation on MNAR data [22, 34]. In IPS, each term of the empirical risk corresponding to an observed rating will be normalized by the inverse of the observation probability (aka propensity). Fortunately, IPS can be directly applied to our latent feature model. We further propose two other debiasing methods specifically designed for our model which unlike IPS does not require explicit propensity scores.

To validate the proposed methods, we present multiple experiments in Section 5, which all confirm the motivating ideas behind the design of the methods, as well as their performances.

In short, our main contributions could be summarized as motivating, formulating, and solving the rating prediction problem as a smooth signal recovery in the frequency domain. This approach offers several advantages over other latent feature models with a similar level of complexity:

- The formulation notably relies on a minimal assumption, i.e., the smoothness of human behavior, enhancing the framework's robustness and applicability to different fields.

- New algorithms are developed to solve the formulated problem. These solutions are empirically evaluated on extensively used benchmarks.

- The solution is characterized by its simplicity and interpretability. For instance, recommendations can be explained in terms of the inner-product frequency-domain representations of users and items. Such simple solutions with near-optimal performance are always favored in machine learning.

- The formulation is flexible, enabling it to address well-documented challenges of data-driven recommender systems (RS). For instance, we demonstrate that by controlling the bandwidth of representations, we can tune the complexity of user and item representations, making it easier to adapt to asymmetry. Additionally, we show that the distribution shift from training to test data can be addressed in new ways, by utilizing frequency-domain representations and insights from the frequency domain. We believe that this frequency representations offer a fresh perspective on the prevalent challenges in recommender systems.

1.1. Related works

The applied methods in CF are versatile and it is difficult to review all of them. Below, we briefly discuss a number of more related methods to our work and what makes our method different from them. A mathematical comparison of the latent feature methods with our model is made in Section 2.

- *MF and its variants.* MF was originally proposed for taking advantage of the inner-product of latent features. The probabilistic version of MF (PMF) [18] further assumes ratings as independent Gaussian random variables. For each rating, the inner-product of latent features determines the mean, and all ratings are assumed to have a similar variance. The maximum likelihood inference in PMF simplifies to the minimization of the mean-squared error of MF predictions with Tikhonov (L_2) regularization. Another improvement to MF was the introduction of row and column bias terms that model systematic biases associated with users and items. This method is commonly known as BiasedMF [14].

- *Deep latent feature methods.* Neural collaborative filtering (NCF) [11] and neural network matrix factorization (NNMF) [7] and are two neural network extensions over MF. Both methods predict ratings using deep neural networks acting on latent features. Our second approach in this work (SmoothRecNet), similarly learns representations for items and users and evaluates the inner-products for the rating prediction. However, the learned representations in our method are based on frequency domain interpretations with controllable complexities. Further, we include an implicit clustering in our network, which significantly improves the performance.

- *Autoencoder-based models.* Autoencoder methods can be seen as models combining aspects of both graph features and latent feature models [28, 24]. They take as input all observed ratings (relations) for a user (entity), allowing the network to implicitly model the graph features, which in this case are similarities among items [7]. We do not study autoencoder-based models here as our focus is on latent feature methods.

- *Spectral CF on graphs.* A recent trend in CF is to investigate the bipartite user-item graph [42, 33] or the user-or-item graph [13] for modeling user-item interactions. These methods use a new spectral convolution filter directly acting in the spectral domain. Although we do not use such graphs, our proposed method benefits from the same key idea: to impose smoothness and to directly learn representations in the frequency domain.

2. Model

Although no two users or items are exactly the same, it is oftentimes possible to fairly represent them with a low-dimensional feature set.

To represent items as vectors, we use the d-dimensional space $\mathbb{R}^d$ (latent item space), where d shall be tuned according to the desired complexity of the representations. Let $\boldsymbol{x}_i \in \mathbb{R}^d$ denote the vector associated to item i, and let $\mathcal{X} \subset \mathbb{R}^d$ be the set of all such $\boldsymbol{x}_i$s.

In general, the rating of a user u to an item i, besides the pair (u, i), can depend on factors such as time or the chronological order with which the user rates the items. Here, we simply assume a time and ordering-independent scoring function. For each user u,

$$h_u : \mathcal{X} \to [s_{\min}, s_{\max}] \tag{1}$$

denotes the scoring function that outputs the rating to any input item. Here, $s_{\min}$ and $s_{\max}$ limit the worst and best possible ratings for an item which depend on the feedback collection system (e.g., $s_{\min} = 1$ and $s_{\max} = 5$).

We show the set of rated and unrated items by the user u with $\mathcal{I}_u^+$ and $\mathcal{I}_u^-$, respectively; this implies that $\mathcal{I}_u^+ \cup \mathcal{I}_u^+ = \mathcal{I}$ for all u, where $\mathcal{I}$ is the set of all items. Similarly, by $\mathcal{U}_i^+$ and $\mathcal{U}_i^-$ we denote the set of users who provided or did not provide a rating for item i, respectively; again, we shall have $\mathcal{U}_i^+ \cup \mathcal{U}_i^- = \mathcal{U}$ for all i, where $\mathcal{U}$ is the set of all users.

With these notations, each observed rating can be expressed as a triplet (u, i, s_{ui}), later called samples, where s_{ui} indicates the value of the rating.

We further denote the set of all available ratings as

$$\begin{aligned}\mathcal{D} &= \{(u, i, s_{ui}) \mid u \in \mathcal{U}, i \in \mathcal{I}_u^+\} \\ &= \{(u, i, s_{ui}) \mid i \in \mathcal{I}, u \in \mathcal{U}_i^+\}.\end{aligned} \tag{2}$$

If item representations $\{\boldsymbol{x}_i\}_i$ are known, the problem of rating prediction can be reduced to the recovery of the scoring function $h_u(\cdot)$, given $\{\boldsymbol{x}_i\}_{i\in\mathcal{I}_u^+}$. But in general, estimating the scoring functions could be formulated as minimizing the loss

$$\mathcal{L}(\{\boldsymbol{x}_i\}_{i\in\mathcal{I}}, \{h_u\}_{u\in\mathcal{U}}) = \sum_{(u,i,s_{ui})\in\mathcal{D}} (s_{ui} - h_u(\boldsymbol{x}_i))^2 + \sum_{u\in\mathcal{U}} \phi(h_u) + \sum_{i\in\mathcal{I}} \psi(\boldsymbol{x}_i) \tag{3}$$

where $\phi : \mathcal{H} \to \mathbb{R}^+$ and $\psi : \mathcal{X} \to \mathbb{R}^+$ are proper regularization terms and $\mathcal{H}$ is the space of all allowed functions. Even with the assumption of known item representations, we need to minimize $\mathcal{L}$ over a function space based on finitely many samples. Without any restrictions on $\mathcal{H}$, $\mathcal{L}$ potentially has infinitely many solutions. In this work, we restrict $\mathcal{H}$ to the *smooth functions*. We will later motivate this selection.

In the sequel, we first show how well-established latent feature methods fit into the current framework (Section 2.1). Then, we motivate the choice of smooth scoring functions (Section 2.2) by a comparison to MF. Although MF is far from state-of-the-art models, we found this comparison insightful. We will then mathematically formulate smooth scoring functions (Section 2.3) and close this section by introducing the optimization problem to recover scoring functions per user (Section 2.4). Algorithms to effectively solve this problem will be the focus of Section 3.

2.1. Scoring functions of other latent feature models

MF and its variants. In the simple yet effective MF method, the users and items are represented in the same vector space $\mathcal{X}$. Let $\boldsymbol{y}_u \in \mathcal{X}$ and $\boldsymbol{x}_i \in \mathcal{X}$ represent user u and item i, respectively. Then, the score of u to i is assumed to have the form

$$h_u^{\mathrm{MF}}(\boldsymbol{x}_i) = \boldsymbol{y}_u^T \boldsymbol{x}_i, \tag{4}$$

where T stands for the transpose operator. A natural extension to this function is to consider per user and per item bias terms to provide more flexibility for incorporating individual discrepancies in the model. MF with added bias is sometimes known as BiasedMF [14] in the literature. In this case, the scoring function of user u follows

$$h_u^{\text{BiasedMF}}(\boldsymbol{x}_i) = \boldsymbol{y}_u^T \boldsymbol{x}_i + \mu + b_u + b_i, \tag{5}$$

where μ, b_u, and b_i are bias terms that are unknown parameters to the learner. A probabilistic view of MF (PMF) [18] assumes independent Gaussian priors over ratings. In this case, the maximum likelihood estimation of latent features is equivalent to the regularized mean-squared error minimization where the regularization terms in Equation 3 are $\phi(\boldsymbol{y}_u) = \lambda\|\boldsymbol{y}_u\|^2$ and $\psi(\boldsymbol{x}_i) = \gamma\|\boldsymbol{x}_i\|^2$.

Neural network matrix factorization (NNMF). In NNMF [7], the inner-product in MF is replaced by a multi-layer feed-forward neural network. The scoring function of user u in this model follows

$$h_u^{\text{NNMF}}(\boldsymbol{x}_i) = f_\theta(\boldsymbol{y}_u \circ \boldsymbol{x}_i, \mu, b_u, b_i), \tag{6}$$

where $\circ$ is the elementwise product and $f_\theta(\cdot)$ is a feed-forward neural network with parameters θ. Note that NNMF is equivalent to MF if $f_\theta(\cdot)$ is the sum operator.

MF with feed-forward multilayer perceptron (MLP). MLP [7], similar to NNMF, the inner-product in MF is replaced with a feed-forward network. Unlike NNMF that acts on $\boldsymbol{y}_u \circ \boldsymbol{x}_i$, MLP gets concatenated $\boldsymbol{y}_u$ and $\boldsymbol{x}_i$ as the input:

$$h_u^{\text{MLP}}(\boldsymbol{x}_i) = \sigma_{out}(\boldsymbol{w}_{out}^T \phi_L(\phi_{L-1}(\cdots\phi_1(\boldsymbol{z}_1)))), \tag{7}$$

where $\boldsymbol{z}_1 = \begin{bmatrix} \boldsymbol{x}_i \\ \boldsymbol{y}_u \end{bmatrix}$, $\phi_l(\boldsymbol{z}_l) = \sigma_l(\boldsymbol{W}_l \boldsymbol{z}_l + \boldsymbol{b}_1)$, and $\sigma_l(\cdot)$ is the activation function of the l^{th} layer. MLP was originally proposed to predict the implicit feedback using $\sigma_{out}(z) = \text{sigmoid}(z)$ and minimizing the cross entropy. In our experiments, we revisit MLP to predict ratings using $\sigma_{out}(z) = z$ and minimizing the mean-squared error. Note that in this method, $\boldsymbol{x}_i$ and $\boldsymbol{y}_u$ are not required to lie in the same vector space.

Neural collaborative filtering (NCF). In this method, MF and MLP learn separate embeddings, and NCF [11] combines the two models by another layer acting on the concatenation of their last hidden layer. Mathematically,

$$h_u^{\text{NCF}}(\boldsymbol{x}_i^{(1)}, \boldsymbol{x}_i^{(2)}) = \sigma\left(\boldsymbol{w}^T \begin{bmatrix} \boldsymbol{x}_i^{(1)} \circ \boldsymbol{y}_i^{(1)} \\ \boldsymbol{z}^{\text{MLP}} \end{bmatrix}\right), \tag{8}$$

where

$$\boldsymbol{z}^{\mathrm{MLP}} = \phi_L(\phi_{L-1}(\cdots \phi_1(\begin{bmatrix} \boldsymbol{x}_i^{(2)} \\ \boldsymbol{y}_u^{(2)} \end{bmatrix}))). \tag{9}$$

2.2. Smooth scoring functions: Motivation

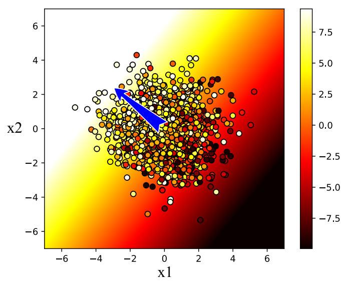
The benchmark matrix factorization (MF) technique is a special case covered by Equation 3; however, it has three main shortcomings: 1) it provides a unimodal scoring, 2) it tends to recommend popular items to everyone, and 3) its performance degrades in cases where the number of items and users differ significantly (asymmetry). We shall show that these shortcomings are special to MF (and its variants) and shouldn't be attributed to the loss minimization framework of Equation 3. In fact, they could be naturally overcome by considering a more general smooth scoring function. This claim will be justified by our experiments (Section 5) on well-studied datasets such as the asymmetric Joke dataset [8]. Below, without going deeply through the algorithms and details of the experiments, we provide some results to motivate the proposed idea.

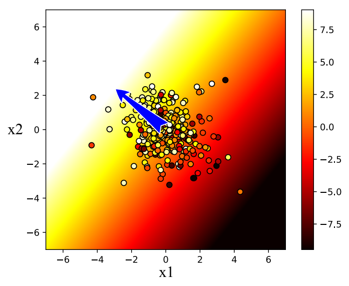
2.2.1. Unimodality vs. multimodality

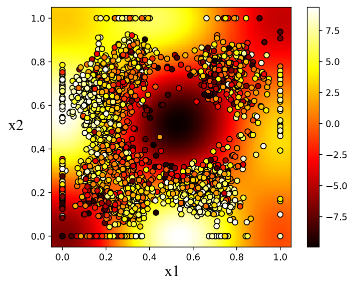
As μ, b_u, and b_i are constant in Equation 5, the scores are modeled linearly in the MF (and in general, BiasedMF) technique. Therefore, $\mathcal{H}^{\mathrm{MF}}$ (class of all h^{MF} functions) consists of monotonic functions with single modes on the edges when we limit the space of $\mathcal{X}$. The background in plots of the first row in Figure 1 show a sample function of $\mathcal{H}^{\mathrm{MF}}$ in 2D. In this figure, items are shown by points with colors determining the actual ratings; different plots correspond to different methods and different training-test data.

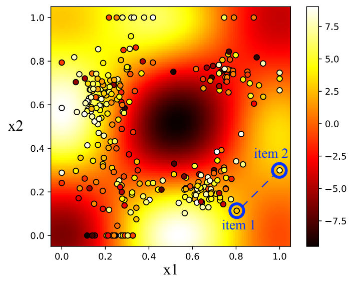
Unimodality of the scoring function forces the modeled user to have only one type of interest, which is oftentimes not true. As an example, in a film rating scenario (items are films), unimodal scoring functions cannot model a user interested in both horror and comedy genres.

In contrast, our proposed class of more general smooth functions ($\mathcal{H}^{\mathrm{smooth}}$) addresses this issue by using multimodal basis. An example function of $\mathcal{H}^{\mathrm{smooth}}$ calculated at $\mathcal{X} = [0, 1)^2$ is shown in the background in plots of the second row in Figure 1. We observe that the estimated smooth scoring function has four modes in this example.


(a) MF (training)


(b) MF (test)


(c) Smooth scoring (training)



(d) Smooth scoring (test)

Figure 1: For a fixed user, the above plots show the actual and predicted scores for a subset of items. In each plot, points (small circles) correspond to 2D representation of the items and their colors reflect the actual scores of the user. The background color-maps represent the predicted scores for each item representation; Figures in the first row, (a) and (b), show the output of MF, centered around $(0,0)$ for the sake of presentation. The output on training and test data are depicted separately. The blue arrow shows the estimated user representation in the MF method, i.e., $\boldsymbol{y}_u$ in Equation 5; Figures in the second row, (c) and (d), correspond to the proposed smooth scoring rule; The same training, validation, and test split are used for both methods.

2.2.2. Popular vs. personalized recommendation

The scoring function in Equation 5 increases by scaling (up or down) the item representation vector $\boldsymbol{x}_i$. This implies that if i is a popular item among most users, its estimated vector $\boldsymbol{x}_i$ is likely to be large in amplitude and well-aligned to most user representations ($\boldsymbol{y}_u$s). The large amplitude of $\boldsymbol{x}_i$ usually dominate the recommendation process overshadowing the in-person differences. Further, due to unimodality, MF pushes popular items in one direction and close to each other, which is not favorable as popular items might represent very different topics. In our proposed method, popular items are not necessarily aligned in one direction, which allows for personal preferences to have greater impact.

2.2.3. Adapting to asymmetric data

In many real recommender systems, the number of users is much higher than the number of items. As an example, consider a telecommunication operator with millions of subscribers in contrast to few hundred SMS/call/internet packages. Recall that in the MF method, users and items shall be represented in the same vector space; in the asymmetric user/item cases, the dimension of the latter vector space is either insufficient for one side (e.g., items) or highly redundant for the other side (e.g., users). As we explain in Section 3, our proposed method has the advantage of using separate vector spaces for users and items with controllable dimensions.

2.2.4. Smooth functions for smooth behavior

As explained earlier, we limit the function space $\mathcal{H}$ in Equation 3 to be a subset of smooth functions. This means that the scores of any user u to items i_1 and i_2 with close representations shall be similar. For instance, items 1 and 2 shown by blue circles in Figure 1d have rather close representations, and we expect their actual ratings (point colors) to be close too. The bandwidth in the class of low-frequency smooth functions $\mathcal{H}^{\text{smooth}}$ (defined in the next subsection) controls the similarity of the scores in terms of the similarity of item representations.

2.3. Smooth scoring functions: Formulation

Before we introduce low-frequency smooth functions, we first study 1D bandlimited signals and their extensions to multi-dimensional functions. Then, we restate $\mathcal{L}$ in Equation 3 for this class of functions.

2.3.1. Frequency-domain representation of 1D bandlimited functions

Let $s[n], n = 0, 1, \ldots, N-1$ be a 1D discrete-domain signal with length N (possibly complex-valued). We say that the bandwidth of s is M ($M < \frac{N}{2}$) if for some complex-valued coefficients $\{a_m\}_{m=-M}^{M}$ we have

$$s[n] = \sum_{m=-M}^{M} a_m \mathrm{e}^{\mathrm{j}\frac{2\pi}{N}m\,n}, \quad \forall\, 0 \leq n < N, \tag{10}$$

where j stands for $\sqrt{-1}$. We can view Equation 10 as linearly relating $2M+1$ *Fourier coefficients* $\{a_m\}_{m=-M}^{M}$ to signal samples $s[n]$. In case we know $2M+1$ distinct samples from s, we can uniquely recover $\{a_m\}_{m=-M}^{M}$ (the resulting linear system of equations is always invertible). It is useful to interpret

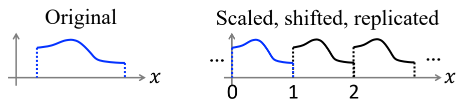
Equation 10 as a discretization of a continuous trigonometric function at $x = \frac{n}{N}$:

$$h(x) = \sum_{m=-M}^{M} a_m \mathrm{e}^{\mathrm{j}2\pi\, m\, x},\ \boldsymbol{a} \in \mathbb{C}^{2M+1},\ x \in [0, 1). \tag{11}$$

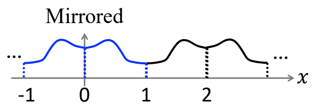
As h is periodic with period $T = 1$, we have limited the input range to $x \in [0, 1)$. Similar to the discrete-domain signal, we call the continuous-domain function $h(x)$ in Equation 11 bandlimited with bandwidth M. With this generalized definition, we can again recover the function $h(x)$ (or its Fourier coefficient vector $\boldsymbol{a}$) by having $r \geq 2M + 1$ samples in the range $[0, 1)$; however, we are no longer restricted to uniformly spaced samples (or samples on a grid in higher dimensions).

2.3.2. From smooth behavior to bandlimited functions

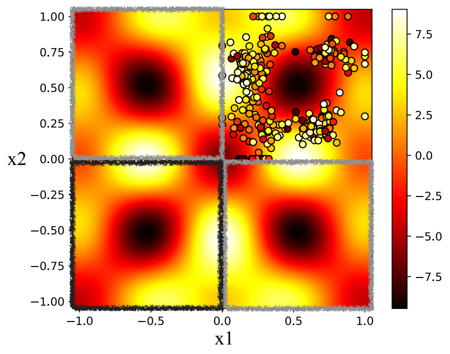
It is well-known that the rate of variations in a function is directly related to its bandwidth. In particular, smooth functions with limited variations can be fairly approximated with low-frequency bandlimited signals. But to make this approximation possible we need two further steps: First, while a bandlimited function as in Equation 11 is smooth and its smoothness can be controlled via M, it is further periodic which is not necessarily the case for the intended scoring functions. Nevertheless, if we assume the scoring functions are defined on a closed domain $\mathcal{X} \in \mathbb{R}^d$, then, they can be extended to periodic forms. For the sake of simplicity, we shift and scale the actual domain to coincide with $\mathcal{X} = [0, 1)^d$. Second, a naive periodic extension of a generic smooth function defined over $\mathcal{X} = [0, 1)^d$ does not necessarily result in a smooth periodic function. Figure 2a illustrates the boundary discontinuities formed in this way. Alternatively, as suggested in Figure 2b, we can first mirror the signal (make it even) and then replicate it to form a periodic function with double the period. As the original function is smooth, this latter periodic extension is also smooth. The same concept is applicable to higher dimensions. Figure 2c illustrates the 2D case. Here we have mirrored the function with respect to each axis and the origin. We then replicate the mirrored function with the period of 2. The resulting function is periodic along both axes and has the desired smooth behavior in the whole domain, so can be fairly approximated by bandlimited functions.



(a)



(b)



(c)

Figure 2: Transforming an arbitrary finite length smooth functions to a periodic smooth function. (a) 1D example. If the original function is scaled to $[0, 1)$ and replicated, it will not necessarily become smooth. (b) If the 1D signal is mirrored before replication, the outcome is both smooth and periodic. (c) 2D example. With $2D$ mirroring and replication, we can achieve a periodic and smooth function; the original signal over $[0, 1)^2$ (quadrant I) is mirrored with respect to the x_2 axis (vertical) in quadrant II, with respect to the x_1 axis (horizontal) in quadrant IV and with respect to both x_1 and x_2 axes in quadrant III. The mirrored signal is then replicated to achieve a periodic smooth signal.

2.3.3. Frequency-domain representation of multi-dimensional real-valued mirrored periodic functions

The mirrored and replicated function explained in Section 2.3.2 is even with respect to all of its inputs. In other words,

$$h(\boldsymbol{x}) = h(x_1, x_2, ..., x_d) = h(\pm x_1, \pm x_2, ..., \pm x_d) \tag{12}$$

for all possible selections of the signs. For such a real-valued even function that has period 2 in each axis, assuming h has the bandwidth of M, Equation 11 can be simplified to

$$h(\boldsymbol{x}) = \sum_{m_1=0}^{M} \cdots \sum_{m_d=0}^{M} \mathsf{A}_{m_1,\cdots,m_d} \big(\prod_{q=1}^{d} \cos(\pi m_q x_q) \big) \tag{13}$$

that consists of only the cosine products. Here, $\boldsymbol{x} \in [0,1)^d$ and A is a d-dimensional real-valued tensor. Given $\boldsymbol{x}$, for all $(M+1)^d$ possible d-tuples $(m_1, \dots, m_d)$ of $\{0, 1, \dots, M\}$ sorted in the lexicographic order, we calculate $\prod_{q=1}^{d} \cos(\pi m_q x_q)$ terms in Equation 13 and stack them in form of a vector $\boldsymbol{v} \in \mathbb{R}^{(M+1)^d}$. Clearly, $\boldsymbol{v}$ is a function of $\boldsymbol{x}$ and should be denoted as $\boldsymbol{v}(\boldsymbol{x})$. Further, let $\boldsymbol{a} \in \mathbb{R}^{(M+1)^d}$ be the vectorized form of A by following the same lexicographic order for the indices of the elements of A. We call $\boldsymbol{a}$ the *Fourier coefficient vector* of h. This allows us to rewrite Equation 13 as

$$h(\boldsymbol{x}) = \boldsymbol{v}^T(\boldsymbol{x})\boldsymbol{a}. \tag{14}$$

2.3.4. Recovery of multi-dimensional functions

For recovering h from non-uniform samples in multi-dimensions, assume we have r samples $\{s_i\}_{i=1}^r$ of h at $\{\boldsymbol{x}_i \in [0,1)^d\}_{i=1}^r$. Let us define the $(M+1)^d \times r$ matrix $\boldsymbol{V}$ as

$$\boldsymbol{V} = [\boldsymbol{v}(\boldsymbol{x}_1), \boldsymbol{v}(\boldsymbol{x}_2), \dots, \boldsymbol{v}(\boldsymbol{x}_r)]. \tag{15}$$

Note that $\boldsymbol{V}$ is a function of $\{\boldsymbol{x}_i\}_{i=1}^r$; however, to simplify the notations we usually drop this dependency.

By adopting such smooth score functions h, we shall have $s_i = h(\boldsymbol{x}_i) = \boldsymbol{v}^T(\boldsymbol{x}_i)\boldsymbol{a}$. Thus, the Fourier coefficient vector can be related to the samples as

$$\boldsymbol{s} = \boldsymbol{V}^T\boldsymbol{a}. \tag{16}$$

Here $\boldsymbol{s}$ is the column vector of r observed s_i values. In contrast to the 1D case, it is not clear if $(M+1)^d$ samples are always sufficient for unique recovery of $\boldsymbol{a}$ in d dimensions when $d > 1$. Indeed, we need as many samples to make sure that the rank of $\boldsymbol{V}$ equals or exceeds $(M+1)^d$. When this happens, we can first recover $\boldsymbol{a}$ based on Equation 16 and then, calculate the output of h for any given input from Equation 14. Putting these together, the scoring function h can be recovered as

$$h(\boldsymbol{x}) = \boldsymbol{v}^T(\boldsymbol{x})(\boldsymbol{V}^T)^\dagger\boldsymbol{s}, \tag{17}$$

where $\dagger$ is the pseudo-inverse operator.

2.4. Per user recovery of the scoring function

Because the scoring functions of different users are not the same, we have to recover h_u (equivalently, $\boldsymbol{a}_u$) for each user u. By rewriting Equation 3 based on Equation 14, we have

$$\mathcal{L}^{\text{smooth}}(\{\boldsymbol{x}_i\}_{i\in\mathcal{I}}, \{\boldsymbol{a}_u\}_{u\in\mathcal{U}}) = \sum_{(u,i,s_{ui})\in\mathcal{D}} \left(s_{ui} - \boldsymbol{v}^T(\boldsymbol{x}_i)\boldsymbol{a}_u\right)^2 + \lambda \sum_{u\in\mathcal{U}} \|\boldsymbol{a}_u\|^2, \quad (18)$$

where we used the L_2 regularizer $\lambda \sum_{u\in\mathcal{U}} \|\boldsymbol{a}_u\|^2$ to make the estimation less prone to overfitting when the number of rated items by a user is not large enough. Now, the optimization problem becomes:

$$\begin{aligned} &\min_{\substack{\{\boldsymbol{x}_i\}_{i\in\mathcal{I}} \\ \{\boldsymbol{a}_u\}_{u\in\mathcal{U}}}} \mathcal{L}^{\text{smooth}} \\ &\text{s.t. } \boldsymbol{x}_i \in [0,1)^d, \ \forall i \in \mathcal{I}. \end{aligned} \quad (19)$$

$\mathcal{L}^{\text{smooth}}$ in Equation 18 is separable in terms of each user's contribution to the loss. For each user u, if we order all $\{s_{ui} \mid i \in \mathcal{I}_u^+\}$ in a column vector $\boldsymbol{s}_u$ and use the same ordering to form $\boldsymbol{V}_u$ from $\{\boldsymbol{v}(\boldsymbol{x}_i)\}_{i\in\mathcal{I}_u^+}$ as in Equation 15, then, Equation 18 can be rewritten as

$$\mathcal{L}^{\text{smooth}}(\{\boldsymbol{x}_i\}_{i\in\mathcal{I}}, \{\boldsymbol{a}_u\}_{u\in\mathcal{U}}) = \sum_{u\in\mathcal{U}} \Big(\|\boldsymbol{s}_u - \boldsymbol{V}_u^T\boldsymbol{a}_u\|^2 + \lambda\|\boldsymbol{a}_u\|^2\Big). \quad (20)$$

We should recall that $\boldsymbol{V}_u$ implicitly depends on item representations that are rated by user u ($\{\boldsymbol{x}_i\}_{i\in\mathcal{I}_u^+}$).

An important property of the loss function in Equation 20 is that if item representations $\{\boldsymbol{x}_i\}_{i\in\mathcal{I}}$ are known, $\mathcal{L}^{\text{smooth}}$ becomes separable with respect to $\boldsymbol{a}_u$; for instance, if λ is small, the optimal Fourier coefficient vectors could be obtained as $\boldsymbol{a}_u \approx (\boldsymbol{V}_u^T)^\dagger \boldsymbol{s}_u$. This property allows us to alternatively tune item representations and Fourier coefficient vectors, as will be explained later in Section 3.

3. Algorithms

The loss function in Equation 20 defines a non-convex minimization problem. Even when item representations are fixed, solving the simpler problem $\boldsymbol{V}_u^T\boldsymbol{a}_u = \boldsymbol{s}_u$ could be problematic due to ill-conditioning of the linear system of equations, especially when the user u has a few recorded ratings.

To reliably estimate the Fourier coefficient vectors $\{\boldsymbol{a}_u\}_{u\in\mathcal{U}}$, to be also called user representations, we group similar users into a number of clusters and use a single representative for each cluster (virtually increasing the number of available ratings). We show these representatives by $\{\boldsymbol{a}_k\}_{k\in\mathcal{C}}$ and call them *cluster (frequency-domain) representations* or *cluster Fourier coefficient vectors*.

It is well-known that grouping users or items into clusters is effective in collaborative filtering [31, 30]; here, we propose a clustering method designed specifically for the recovery of smooth scoring functions. In this approach, we modify Equation 20 as

$$\mathcal{L}_*^{\text{smooth}}(\{\boldsymbol{x}_i\}_{i\in\mathcal{I}}, \{\boldsymbol{a}_k\}_{k\in\mathcal{C}}, c) = \sum_{u\in\mathcal{U}} \|\boldsymbol{s}_u - \boldsymbol{V}_u^T \boldsymbol{a}_{c(u)}\|^2 + \lambda \sum_{k\in\mathcal{C}} \|\boldsymbol{a}_k\|^2, \qquad (21)$$

where $\mathcal{C}$ is the set of clusters, $c : \mathcal{U} \to \mathcal{C}$ is the mapping of the users into clusters, and the subscript $*$ is added to distinguish between the clustered and non-clustered (Equation 20) loss functions. It is easy to see that the non-clustered loss function defined in Equation 20 is a special case of the clustered loss function defined in Equation 22, by assuming each user as a separate cluster ($c(u) = u$). Note that again $\boldsymbol{V}_u$ in $\mathcal{L}_*^{\text{smooth}}$ implicitly depends both on item representations and the items rated by each user. In addition, the regularization parameter λ shall be tuned via cross-validation. The optimization problem corresponding to this clustered loss function is

$$\begin{aligned} &\min_{\substack{\{\boldsymbol{x}_i\}_{i\in\mathcal{I}} \\ \{\boldsymbol{a}_k\}_{k\in\mathcal{C}} \\ c(\cdot)}} \mathcal{L}_*^{\text{smooth}} \\ &\text{s.t. } \ \boldsymbol{x}_i \in [0,1)^d, \quad \forall i \in \mathcal{I}. \end{aligned} \qquad (22)$$

In the sequel, we introduce two approaches for solving the optimization problem of Equation 22. First, we propose an alternating optimization method with a similar formulation as in the alternating least-squares (ALS) method previously applied to MF [14]. There, we introduce *k-representation*, a clustering method inspired by k-means. Second, we propose a shallow feed-forward neural network named *SmoothRecNet*, which concurrently learns item representations, groups users into soft clusters, and learns cluster representations. Finally, we discuss how to integrate a user-based model to the problem, as represented in our current formulation, with an item-based approach.

3.1. Alternating optimization

Alternating optimization is an iterative technique in which the variables of the loss function are divided into a number of groups and at each iteration, the loss function is minimized over the variables in one of the groups while the rest are assumed fixed. Thereafter, the variable group cyclically alters within consecutive iterations. In this regard, we break down the optimization problem of Equation 22 into the following steps:

- *Update item representations*: By keeping the clusters and cluster Fourier coefficient vectors fixed, we minimize $\mathcal{L}_*^{\text{smooth}}$ w.r.t. item representations $\{\boldsymbol{x}_i\}_{i\in\mathcal{I}}$. The details are covered in Section 3.1.1.
- *Update clusters and cluster Fourier coefficient vectors*: For fixed item representations and consequently fixed $\{\boldsymbol{V}_u\}_{u\in\mathcal{U}}$, we find clusters, i.e., the mapping $c(\cdot)$ with associated cluster Fourier coefficient vectors $\{a_k\}_{k\in\mathcal{C}}$, that minimize $\mathcal{L}_*^{\text{smooth}}$. The details are provided in Section 3.1.2.

3.1.1. Updating item representations

The first part of the alternating optimization is to minimize $\mathcal{L}_*^{\text{smooth}}$ w.r.t. item representations $\{\boldsymbol{x}_i\}_{i\in\mathcal{I}}$ for a fixed $c(\cdot)$ and $\{\boldsymbol{a}_k\}_{k\in\mathcal{C}}$. To simplify the equations, we separate $\mathcal{L}_*^{\text{smooth}}$ of Equation 21 as a sum over items:

$$\mathcal{L}_*^{\text{smooth}}(\{\boldsymbol{x}_i\}_{i\in\mathcal{I}}, \{\boldsymbol{a}_k\}_{k\in\mathcal{C}}, c) = \sum_{i\in\mathcal{I}} \Big(\sum_{u\in\mathcal{U}_i^+} l_*^{\text{smooth}}(\boldsymbol{x}_i, \boldsymbol{a}_{c(u)}, s_{ui}) \Big) + \lambda \sum_{k\in\mathcal{C}} \|\boldsymbol{a}_k\|^2, \tag{23}$$

where

$$l_*^{\text{smooth}}(\boldsymbol{x}, \boldsymbol{a}, s) = \big(s - \boldsymbol{v}^T(\boldsymbol{x})\boldsymbol{a}\big)^2. \tag{24}$$

This lets us update item representations independently. We use *L-BFGS-B* algorithm [2, 43] from the `scipy` python package as the optimization algorithm. The L-BFGS-B is an approximation of the BFGS algorithm from the family of quasi-newton methods with limited memory. This algorithm uses only the first derivative and has a suitable performance even in non-smooth optimizations. It can further handle simple box constraints like $\boldsymbol{x}_i \in [0,1)^d$. In this technique, we need to calculate the gradient of l_*^{smooth} w.r.t. $\boldsymbol{x}_i$. It is evident that $\nabla_{\boldsymbol{x}_j} l_*^{\text{smooth}}(\boldsymbol{x}_i, \boldsymbol{a}_{c(u)}, s_{ui})$ is zero for all $j \neq i$. For $q = 1, 2, \ldots, d$

we have

$$\frac{\partial}{\partial x_{i,q}} l_*^{\text{smooth}}(\boldsymbol{x}_i, \boldsymbol{a}_{c(u)}, s_{ui}) =$$
$$2\big(s_{ui} - \boldsymbol{v}^T(\boldsymbol{x}_i)\boldsymbol{a}_{c(u)}\big) \sum_{n=1}^{(M+1)^d} \pi a_{c(u),n}\, v_n(\boldsymbol{x}_i)\, m(n,q)\, \tan\big(\pi m(n,q) x_{i,q}\big), \quad (25)$$

where $m(n,q)$ is m_q in the nth d-tuple $(m_1, \ldots, m_d)$ after lexicographic sorting (see Section 2.3.3). An efficient implementation of this equation with matrix multiplications is provided in our open-source python package.[2]

Pre-search. Before using L-BFGS-B, we make use of the current function evaluations to update item representations. Consider we are in the t^{th} iteration and the values of $\{\boldsymbol{v}(\boldsymbol{x}_i^{(t-1)})\}_{i\in\mathcal{I}}$ and $\{\boldsymbol{a}_u^{(t-1)} = \boldsymbol{a}_{c^{(t-1)}(u)}^{(t-1)}\}_{u\in\mathcal{U}}$ from the previous iteration are available. For every $i \in \mathcal{I}$, define

$$i^+ = \arg\min_{j\in\mathcal{I}} \sum_{u\in\mathcal{U}_i^+} l_*^{\text{smooth}}(\boldsymbol{x}_j, \boldsymbol{a}_u, s_{u,i}). \quad (26)$$

Based on the representation of item i^+ for item i, let us define

$$\boldsymbol{x}_i^{(t-\frac{1}{2})} = \boldsymbol{x}_{i^+}^{(t-1)}. \quad (27)$$

Then we use $\boldsymbol{x}_i^{(t-\frac{1}{2})}$ as the input to L-BFGS-B and run a few iterations to get $\boldsymbol{x}_i^{(t)}$. This pre-search makes the optimization process less prone to stagnation in local minima.

3.1.2. Updating clusters and cluster Fourier coefficient vectors

We propose *k-representation* (Algorithm 1) to find clusters and their representations iteratively. In short, we first draw initial cluster representations $\{\boldsymbol{a}_k\}_{k\in\mathcal{C}}$ from a Gaussian distribution. Then, we assign each user to the cluster that minimizes the user's contribution to the total loss. After assigning users to clusters, we update each cluster representation $\boldsymbol{a}_k$, using

$$\boldsymbol{a}_k = (\boldsymbol{V}_k \boldsymbol{V}_k^T + \lambda \boldsymbol{I})^{-1} \boldsymbol{V}_k \boldsymbol{s}_k. \quad (28)$$

Here $\boldsymbol{s}_k$ is a column vector obtained by stacking column vectors $\{\boldsymbol{s}_u\}_{u:c(u)=k}$, and $\boldsymbol{V}_k$ is the concatenation of $\{\boldsymbol{V}_u\}_{u:c(u)=k}$ in the second dimension. Hence,

[2] `https://github.com/alishiraliGit/collaborative-filtering-in-the-frequency-domain`

Algorithm 1 *k-representation*

```
input:
    item representations $\{\boldsymbol{x}_i\}_{i \in \mathcal{I}}$
    observed ratings $\mathcal{D}$
    number of clusters $|\mathcal{C}|$
    initialization variance of cluster representations $\sigma^2$
    $L_2$ penalty parameter $\lambda$.
output:
    user to cluster mapping $c : \mathcal{U} \to \mathcal{C} = \{1, \dots, |\mathcal{C}|\}$
    cluster representations $\{\boldsymbol{a}_k\}_{k \in \mathcal{C}}$
procedure $k$-REPRESENTATION
    for all $u \in \mathcal{U}$ do calculate $\boldsymbol{V}_u$ from $\{\boldsymbol{x}_i\}_{i \in \mathcal{I}_u^+}$
    for all $k \in \mathcal{C}$ do draw $\boldsymbol{a}_k^{(0)}$ from $\mathcal{N}(0, \sigma^2)$
    repeat
        for all $u \in \mathcal{U}$ do $c^{(t+1)}(u) \leftarrow \mathrm{argmin}_k \|\boldsymbol{s}_u - \boldsymbol{V}_u^T \boldsymbol{a}_k^{(t)}\|^2$
        for all $k \in \mathcal{C}$ do calculate $\boldsymbol{V}_k^{(t+1)}$ by concatenating $\{\boldsymbol{V}_u\}_{u:c^{(t+1)}(u)=k}$
        for all $k \in \mathcal{C}$ do update $\boldsymbol{a}_k^{(t+1)}$ via Equation 28 given $\boldsymbol{V}_k^{(t+1)}$
    until convergence
    return $\{\boldsymbol{a}_k^{(\mathrm{end})}\}_{k \in \mathcal{C}}$ and $c^{(\mathrm{end})}(.)$
end procedure
```

$\boldsymbol{a}_k$ is the solution to the over-determined linear system of equations $\boldsymbol{s}_k = \boldsymbol{V}_k^T \boldsymbol{a}_k$ with L_2 regularization.

We repeat this procedure multiple times to reach a stationary point. Similar to the k-means clustering algorithm, we cannot establish any theoretical guarantee for the convergence of k-representation to the global minimizer; nevertheless, the overall loss is decreased in each iteration. We shall evaluate the performance of the method in Section 5 on both synthetic and real data.

Boosted k-representation. By introducing both the clustering and L_2 regularization in Equation 28, we have improved the robustness of the inverse problem. However, increasing the number of clusters is still a potential issue in estimating the cluster Fourier coefficient vectors. Here, we propose to learn an ensemble of weak binary clusterings instead of learning all clusters together (Algorithm 2). The idea is to find the residuals of the predicted ratings for each user and fit a new clustering to the residuals. Due to the linearity of the predictions, the final representation for user u, $\boldsymbol{a}_u$, will be the

Algorithm 2 *boosted k-representation*

```
input:
    as in Algorithm 1 + number of weak clusterings ($L$)
output:
    user representations $\{\boldsymbol{a}_u\}_{u\in\mathcal{U}}$
procedure BOOSTED $k$-REPRESENTATION
    for all $u \in \mathcal{U}$ do $\boldsymbol{a}_u \leftarrow 0,\ ,\ \boldsymbol{s}_u^{(0)} \leftarrow \boldsymbol{s}_u$
    for $l = 1 : L$ do
        $\{\boldsymbol{a}_k^{(l)}\}_{k\in\{0,1\}}, c^{(l)}(\cdot) \leftarrow$ $k$-rep. with 2 clusters and $\{\boldsymbol{s}_u^{(l-1)}\}$ as ratings
        for $u \in \mathcal{U}$ do
            $\boldsymbol{a}_u^{(l)} \leftarrow \boldsymbol{a}_{c^{(l)}(u)}^{(l)}$
            $\boldsymbol{s}_u^{(l)} \leftarrow \boldsymbol{s}_u^{(l-1)} - \boldsymbol{V}_u^T \boldsymbol{a}_u^{(l)}$
            $\boldsymbol{a}_u \leftarrow \boldsymbol{a}_u + \boldsymbol{a}_u^{(l)}$
        end for
    end for
    return $\{\boldsymbol{a}_u\}_{u\in\mathcal{U}}$
end procedure
```

sum of cluster representations of the weak clusters obtained so far to which u belongs.

3.1.3. Time complexity of the alternating optimization

In updating item representations, for each item i, accessing an element of the gradient $\nabla_{\boldsymbol{x}_i} l_*^{\text{smooth}}$ requires $(M+1)^d$ operations. The time complexity of a single iteration of the BFGS method is dominated by the operations needed to update the inverse Hessian matrix and the matrix-vector products, which typically take $\mathcal{O}(d^2)$ operations in a d-dimensional space. In practice, however, we run L-BFGS-B only for a few iterations per alternation and for a couple of alternations; therefore, the overall time complexity of updating item representations is $\mathcal{O}\big((M+1)^d|\mathcal{I}|d^2\big)$.

For finding the clusters, assigning user u to a cluster requires solving $\text{argmin}_k\|\boldsymbol{s}_u - \boldsymbol{V}_u^T\boldsymbol{a}_k^{(t)}\|^2$. If there are n_u training data per user u, this minimization requires $\mathcal{O}\big(n_u(M+1)^d|\mathcal{C}|\big)$ operations. Hence, updating the clusters for all users requires $\mathcal{O}\big((M+1)^d|\mathcal{C}|\,|\mathcal{D}|\big)$ operations, where $|\mathcal{D}|$ is the size of training data. After assigning users to clusters, the matrix inversion in Equation 28 for updating cluster representations is the computational bottleneck, which needs $\mathcal{O}\big((M+1)^{3d}\big)$ operations for a vanilla solver. Thus, overall, updating clusters and cluster representations has a complexity of $\mathcal{O}\big((M+1)^d|\mathcal{C}|\,|\mathcal{D}|+(M+1)^{3d}|\mathcal{C}|\big)$. If boosted with L weak binary learners, this complexity reduces to $\mathcal{O}\big((M+1)^d|\mathcal{D}|+(M+1)^{3d}L\big)$.

As our experiments on real-world datasets show, small values of d are often sufficient to represent the item space. Therefore, the exponential dependence on d is less of a concern.

3.2. SmoothRecNet: A shallow feed-forward network for learning smooth scoring functions

Recent advances in deep learning have made neural networks great tools even for traditional well-known algebraic calculations. Specifically, neural networks can be optimized effectively with various methods and efficient implementations that boost the training. Further, some useful techniques such as batch normalization, drop-out, etc., are available to avoid overfitting. In this section, we reformulate $\mathcal{L}_*^{\text{smooth}}$ of Equation 21 to be based on soft clusters instead of hard clusters and design an architecture that learns item and user representations concurrently while predicting ratings.

3.2.1. Soft clustering

The vector-valued assignment function $\boldsymbol{c}:\mathcal{U}\rightarrow\mathbb{R}^{|\mathcal{C}|}$ determines to what extent a user belongs to each cluster; therefore, it can be interpreted as a soft clustering. The output of $\boldsymbol{c}(\cdot)$ for the user u is a vector such that its k^{th} element, shown by $c_k(u)$, indicates u's association to cluster k. We refrain from constraining the norm of $\boldsymbol{c}$ and let it scale appropriately for different users. Using this vector-valued assignment function, we modify the loss in Equation 21 as

$$\mathcal{L}_{\text{soft}}^{\text{smooth}}(\{\boldsymbol{x}_i\}_{i\in\mathcal{I}},\{\boldsymbol{a}_k\}_{k\in\mathcal{C}},\boldsymbol{c}) = \sum_{u\in\mathcal{U}}\big\|\boldsymbol{s}_u-\sum_{k\in\mathcal{C}}c_k(u)\;\boldsymbol{V}_u^T\boldsymbol{a}_k\big\|^2+\lambda\sum_{k\in\mathcal{C}}\|\boldsymbol{a}_k\|^2. \tag{29}$$

Figure 3 shows an architecture inspired by Equation 29. It consists of three trainable layers (namely *X-layer*, *soft-clustering-layer*, and *user-layer*). The observed ratings are supplied per item. Each neuron at the input-layer (*item-one-hot-layer*) corresponds to an item, and the input vectors shall be of one-hot form[3]. Next, we have the *X-layer*, which is a dense layer with d units. We interpret the weights from unit i of the input layer to the X-layer as $\boldsymbol{x}_i$. Before the *V-layer* there are few non-trainable layers functioning as $\prod_{q=1}^{d} \cos(\pi m_q x_q)$, so we see $\boldsymbol{v}(\boldsymbol{x}_i)$ at the output of the V-layer[4]. The layer after the V-layer is the last hidden layer with n_1 neurons which we call the *soft-clustering-layer*. We interpret the weights going from the V-layer to unit k of the soft-clustering-layer as $\boldsymbol{a}_k$, i.e., the Fourier coefficient vector of the k^{th} soft cluster. Then, the value of each unit in the soft-clustering-layer will be $\boldsymbol{v}^T(\boldsymbol{x}_i)\boldsymbol{a}_k$, which is the predicted rating for that cluster. Finally, the output-layer or equivalently the *user-layer* is a dense layer with $|\mathcal{U}|$ units. Weights going from the soft-clustering-layer to unit u of the user-layer determine $\boldsymbol{c}(u)$. We shall justify this interpretation using a synthetic dataset in Section 5.

The drop-out layer (not depicted) after the V-layer has an important role in preventing the network from overfitting. The drop-out technique is justifiable because the observed ratings usually contain a lot of uncertainty in real applications; for instance, a user might rate the same item differently when asked twice. The drop-out layer prevents the model from overfitting by stopping the network from relying too much on a specific part of the input data.

3.2.2. Adding non-linearity

One way to increase the capacity of our model is to let the user scoring functions depend non-linearly on the predicted cluster ratings. For this purpose, we rewrite Equation 29 as

$$\begin{aligned}&\mathcal{L}^{\text{smooth}}_{\text{nonlinear}}(\{\boldsymbol{x}_i\}_{i\in\mathcal{I}}, \{\boldsymbol{a}_k\}_{k\in\mathcal{C}}, \boldsymbol{c}) = \\ &\quad \sum_{(u,i,s_{ui})\in\mathcal{D}} \big(s_{ui} - g_u(\{\boldsymbol{v}^T(\boldsymbol{x}_i)\boldsymbol{a}_k\}_{k\in\mathcal{C}})\big)^2 + \lambda \sum_{k\in\mathcal{C}} \|\boldsymbol{a}_k\|^2,\end{aligned} \tag{30}$$

where $g_u : \mathbb{R}^{|\mathcal{C}|} \to \mathbb{R}$ is any function that combines the predicted ratings of $|\mathcal{C}|$ clusters to predict the rating of user u. In Figure 3, we have proposed

[3]Note that extending the input-layer with further neurons enables us to model side information; nevertheless, we ignore any side information for now.

[4]Autograd engines do not require explicit layers and structures as we have provided in Figure 3. But to make our approach implementable within all popular deep learning frameworks, we have demonstrated our design based on typically available layers.

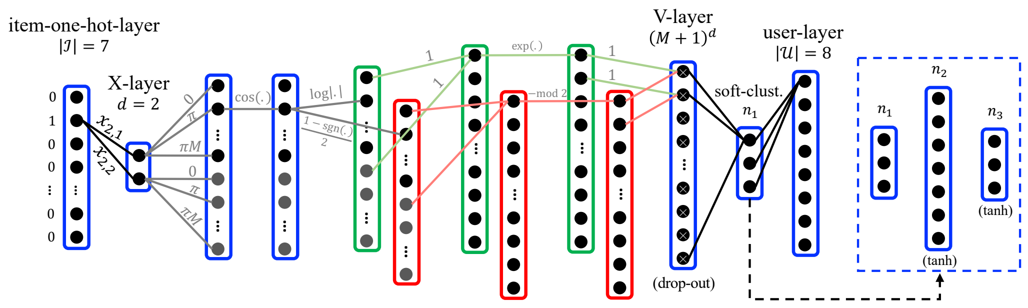


Figure 3: A neural network inspired by Equation 29. The names of the layers are written on top. The value below a layer name shows the number of neurons in that layer. Given the one-hot representation of an item i, we first extract its representation, $\boldsymbol{x}_i$, in the X-layer. Then, we use additional layers (red and green boxes) to construct the product of cosine terms in the definition of $\boldsymbol{v}(\boldsymbol{x}_i)$. The V-layer outputs $\boldsymbol{v}(\boldsymbol{x}_i)$ and the outputs of the following layer act as soft-clustering results. Finally, the predicted ratings for all users appear in the user-layer. The blue box in the right part of the figure shows the non-linear extension of user representations explained in Section 3.2.2.

the implementation of $g_u(\cdot)$ using two additional hidden layers in the right panel with tanh activation function. We usually set $n_1 = n_3$ matching the number of soft clusters in the data. Let $\boldsymbol{W}_2$ and $\boldsymbol{W}_3$ be the weights of these two dense hidden layers, mathematically, $g_u(\cdot)$ can be written as

$$g_u(\{\boldsymbol{v}_i^T \boldsymbol{a}_k\}_{k\in\mathcal{C}}) = \sum_{q\in\widetilde{\mathcal{C}}} \widetilde{c}_q(u)\, \widetilde{g}_q(\boldsymbol{W}_2, \boldsymbol{W}_3, \{\boldsymbol{v}_i^T \boldsymbol{a}_k\}_{k\in\mathcal{C}}), \tag{31}$$

where $|\widetilde{\mathcal{C}}| = n_3$. In this equation, we interpret $\widetilde{\mathcal{C}}$ as the set of super-clusters that non-linearly combine the output of clusters in $\mathcal{C}$. Then, $\widetilde{\boldsymbol{c}} : \mathcal{U} \to \mathbb{R}^{|\widetilde{\mathcal{C}}|}$ will be the (soft) assignment function of the users to super-clusters. With this formulation, one can still interpret the weights going from the last hidden layer to the user-layer as the inclusion of those users in super-clusters.

3.3. Combining user-based and item-based predictors

So far we have assumed that items are mapped to a vector space, and users have scoring functions (user-based method). However, it is also possible to consider to consider it the other way around: items score users (item-based method). A simple way of combining user-based and item-based methods is to do a linear regression of the predictions made by both methods to the observed ratings. This linearity preserves the interpretability of the representations and simplicity of our approach. The validation part of the data will serve for estimating the coefficients of the regression. We observed that combining user-based and item-based methods significantly improves the quality of predictions.

4. Mitigating the effect of missing-not-at-random observations

Observed ratings are often collected using another RS called the *logging* RS. Since the logging RS aims at recommending favorable items, we expect the available ratings to be no longer randomly observed and oftentimes with a higher average compared to unobserved ratings [26]. In addition, even when the logging RS had no intention of offering favorable items, users have a tendency to provide feedback only for items they liked. The combination of these two phenomena result in distribution mismatch between the available and unavailable ratings, which is called in the literature as data with missing-not-at-random (MNAR) [16].

In this section, we address the challenge of missing-not-at-random observations in three different ways. First, we shall explain how the popular method of inverse propensity scoring can be applied to our method. Powered by the simplicity of our formulation, we then propose an effective bias correction method that can be applied without any further training and explicit propensity scores. Finally, we shall briefly discuss how we can leverage frequency-domain intuitions to obtain a frequency-aware regularization as a mean of filtering. Before further explaining the methods, we explicitly define the model.

Model. Let $\mathcal{O} = \{(u,i) \mid u \in \mathcal{U}, i \in \mathcal{I}_u^+\}$ be the set of user-item pairs with observed ratings. Define observation matrix $\boldsymbol{O} \in \{0,1\}^{|\mathcal{U}|\times|\mathcal{I}|}$ with $O_{ui} = 1$ if and only if $(u,i) \in \mathcal{O}$. O_{ui} is a random variable with *propensity score* $P_{ui} = \Pr(O_{ui} = 1) = \mathbb{E}[O_{ui}]$. Generally, P_{ui} depends on the rating s_{ui} itself and side information $\boldsymbol{z}_u$ and $\boldsymbol{z}_i$ from user u and item i that was available to the logging RS. Depending on our knowledge about the logging RS, two settings can be distinguished: i) in the *experimental* setting, the mechanism of the logging RS and consequently propensity scores are known, while ii) in the *observational* setting, propensity scores should be estimated. Unless otherwise stated, we assume the observational setting. Note that in the absence of side information, we need to estimate P_{ui} as a function of s_{ui}.

4.1. Inverse propensity scoring (IPS)

Given a set of true and estimated ratings $\{s_{ui}\}$ and $\{\hat{s}_{ui}\}$, respectively, we are interested in the total loss over *all* possible user-item ratings:

$$\mathcal{L}^*(\{\hat{s}_{ui}\}) = \sum_{u\in\mathcal{U}, i\in\mathcal{I}} l(s_{ui}, \hat{s}_{ui}). \tag{32}$$

Here $l(\cdot,\cdot)$ is the specific form of the loss (e.g., the squared error $(s_{ui} - \hat{s}_{ui})^2$). Unfortunately, $\mathcal{L}^*$ cannot be evaluated as observing s_{ui} for $(u, i) \notin \mathcal{O}$ requires further experimentation. Instead, we can hope the total loss on a *holdout* subset $\mathcal{D}_{\text{test}}$ from the observed data is a fair representative:

$$\mathcal{L}^{\text{test}}(\{\hat{s}_{ui}\}) = \sum_{(u,i,s_{ui})\in\mathcal{D}_{\text{test}}} l(s_{ui}, \hat{s}_{ui}) = \sum_{u\in\mathcal{U}, i\in\mathcal{I}} O_{ui}\, T_{ui}\, l(s_{ui}, \hat{s}_{ui}). \tag{33}$$

Here, T_{ui} is a binary random variable indicating whether the rating from user u to item i is included in the test set given this rating is observed. We assume this inclusion is completely random: $T_{ui} \overset{\text{i.i.d.}}{\sim} \text{Bernoulli}(q)$. We shall now discuss how O_{ui} is distributed. Under missing completely at random assumption $O_{ui} \overset{\text{i.i.d.}}{\sim} \text{Bernoulli}(p)$. In this case, $\mathbb{E}[\mathcal{L}^{\text{test}}(\{\hat{s}_{ui}\})] = pq\mathcal{L}^*(\{\hat{s}_{ui}\})$; therefore, $\mathcal{L}^{\text{test}}$ is a scaled version of $\mathcal{L}^*$ in expectation[5], and can be used as a reliable measure of prediction performance[6].

With missing not at random data, in general s_{ui} can affect O_{ui} and $\mathbb{E}[\mathcal{L}^{\text{test}}(\{\hat{s}_{ui}\})]$ will be a distorted version of $\mathcal{L}^*(\{\hat{s}_{ui}\})$. However, given accurate non-zero propensity scores, there is still a simple fix called inverse propensity scoring (IPS):

$$\mathcal{L}^{\text{IPS,test}}(\{\hat{s}_{ui}\}) = \sum_{(u,i,s_{ui})\in\mathcal{D}_{\text{test}}} \tfrac{1}{P_{ui}} l(s_{ui}, \hat{s}_{ui}). \tag{34}$$

[5]There is a subtlety taking the expectation here. In fact, estimated ratings depend on the part of data that are observed, and one might expect to see the randomness of O_{ui} to exist in $\hat{s}_{ui}$ as well. This could be true if the same set of ratings was used for training and test. To see this, note that the summand in L^{test} is only nonzero if $T_{ui} = 1$. But this means that the data is included in the test, thus, $O_{ui} \perp\!\!\!\perp \hat{s}_{ui} | T_{ui} = 1$.

[6]It is common to interpret the performance on the test data as the true performance. However, there is always a possibility of information leak from the test set after many evaluations on the same set, known as adaptation. This is beyond the scope of our study and is extensively discussed in other works [6, 26].

It is straightforward to see that the IPS estimator is simply a scaled version of $\mathcal{L}^*$[7]:

$$\mathbb{E}[\mathcal{L}^{\text{IPS,test}}(\{\hat{s}_{ui}\})] = \mathbb{E}\Big[\sum_{u\in\mathcal{U},i\in\mathcal{I}} \tfrac{O_{ui}}{P_{ui}} T_{ui} l(s_{ui},\hat{s}_{ui})\Big]$$
$$= \sum_{u\in\mathcal{U},i\in\mathcal{I}} \tfrac{\mathbb{E}[O_{ui}]}{P_{ui}} q\, l(s_{ui},\hat{s}_{ui}) = q\,\mathcal{L}^*(\{\hat{s}_{ui}\}). \tag{35}$$

Thus far, we defined the IPS estimator on the test data. We can similarly weight the training data by the inverse propensity scores and reduce the effect of not-at-random missing data on the obtained predictor. Particularly, neglecting the regularization terms, IPS can be similarly applied to all of the loss functions previously defined. For example, IPS-corrected version of $\mathcal{L}^{\text{smooth}}$ in Equation 18 is given by

$$\mathcal{L}^{\text{smooth,IPS}}(\{\boldsymbol{x}_i\}_{i\in\mathcal{I}},\{\boldsymbol{a}_u\}_{u\in\mathcal{U}}) = \sum_{(u,i,s_{ui})\in\mathcal{D}} \tfrac{1}{P_{ui}}\big(s_{ui} - \boldsymbol{v}^T(\boldsymbol{x}_i)\boldsymbol{a}_u\big)^2. \tag{36}$$

IPS is known to suffer from high variance. Techniques like propensity clipping [27], doubly-robust estimation [5], and self-normaliztion [29] can be applied to further improve the IPS.

4.2. Test-time bias correction

IPS provides a way to correct any loss function through reweighting of the samples. If applied during training, it can indirectly correct the obtained predictor. But the simplicity of our formulation allows a more direct way to correct the obtained predictor without retraining it which unlike IPS does not require explicit propensity scores. We call the proposed method *test-time bias correction* and explain it below.

We shall correct the bias per user. Hence, for a fixed user u, we drop all subscripts u to simplify the notation. Assume that the ratings of user u follow the mentioned smooth form of Equation 14 with the bandwidth M^* and Fourier coefficient vector $\boldsymbol{a}^*$, except for an additive noise:

$$s_i = \boldsymbol{v}_{M^*}^T(\boldsymbol{x}_i)\boldsymbol{a}^* + n_i. \tag{37}$$

We assume n_is are independent and identically distributed (i.i.d.) with a zero mean and variance σ_n^2.

[7]In fact, normalizing the IPS estimator by a factor of q results in an unbiased estimator.

For fixed item representations $\{\boldsymbol{x}_i\}_{i\in\mathcal{I}}$, we want to recover $\boldsymbol{a}^*$ from observed ratings. To do so, we use smooth functions of bandwidth M and minimize the contribution of the user in $\mathcal{L}^{\text{smooth}}$ of Equation 18:

$$l(\boldsymbol{a}) = \sum_{i\in\mathcal{I}^+}(\boldsymbol{v}_M^T(\boldsymbol{x}_i)\boldsymbol{a} - \boldsymbol{v}_{M^*}^T(\boldsymbol{x}_i)\boldsymbol{a}^* - n_i)^2 + \|\boldsymbol{C}\boldsymbol{a}\|^2. \tag{38}$$

Here $\boldsymbol{C}$ is a diagonal weight matrix[8]. We further assume M is sufficiently large, i.e., $M \geq M^*$. By adding zero elements to $\boldsymbol{a}^*$ for higher frequencies, we can use $\boldsymbol{v}_M$ instead of $\boldsymbol{v}_{M^*}$ in Equation 37. For simplicity, we still show the zero-padded $\boldsymbol{a}^*$ as $\boldsymbol{a}^*$ and drop the subscript M from $\boldsymbol{v}_M$. With these conventions, we can simplify Equation 38 as

$$l(\boldsymbol{a}) = \sum_{i\in\mathcal{I}^+}(\boldsymbol{v}^T(\boldsymbol{x}_i)\Delta\boldsymbol{a} - n_i)^2 + \|\boldsymbol{C}\boldsymbol{a}\|^2, \tag{39}$$

where $\Delta\boldsymbol{a} = \boldsymbol{a} - \boldsymbol{a}^*$. Since $l(\boldsymbol{a})$ is smooth and convex in $\boldsymbol{a}$, we can find its minimizer using $\nabla_{\boldsymbol{a}} l = 0$.Define $\hat{\boldsymbol{K}^+} = \sum_{i\in\mathcal{I}^+}\boldsymbol{v}(\boldsymbol{x}_i)\boldsymbol{v}^T(\boldsymbol{x}_i)$. If the regularization term is sufficiently large so that $(\hat{\boldsymbol{K}^+} + \boldsymbol{C})$ is not singular, it is easy to verify that $\hat{\boldsymbol{a}}$ below solves the first-order condition:

$$\hat{\boldsymbol{a}} = (\hat{\boldsymbol{K}^+} + \boldsymbol{C})^{-1}\hat{\boldsymbol{K}^+}\boldsymbol{a}^* + (\hat{\boldsymbol{K}^+} + \boldsymbol{C})^{-1}\Big(\sum_{i\in\mathcal{I}^+}\boldsymbol{v}(\boldsymbol{x}_i)n_i\Big). \tag{40}$$

In a standard ridge regression setting, the second term of the right-hand-side called the noise term, is centered around zero and often neglected. But this might not be the case when $\mathcal{I}^+$ is not a random subset of $\mathcal{I}$. Particularly, for larger s_is we expect to have larger n_is. This implies that the noise term might add an unwanted bias to our estimation. Next, we first formalize this intuition by modeling the first-order effect of s_i on the observation probability. Then, we explain how to estimate the model parameters and correct $\hat{\boldsymbol{a}}$ by subtracting the estimated noise term.

4.2.1. Missing data model

In a missing completely at random setting, $P_i = \Pr(O_i = 1) = p_0$ for all items. But in a missing not at random setting, P_i can depend on s_i. We

[8]$\boldsymbol{C}$ is the identity matrix in Equation 18 but in general we can use different weights for each component of $\boldsymbol{a}$.

study the first-order approximation

$$P_i \approx p_0(1+\alpha^*(s_i-\mu^*)), \tag{41}$$

where $\mu^* = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} s_i$ and $\alpha^* \ll 1$ by assumption. Next, we discuss how to estimate μ^* and α^*.

4.2.2. Estimating missing data model parameters

We estimate μ^* by approximating $s_i \approx \boldsymbol{v}^T(\boldsymbol{x}_i)\hat{\boldsymbol{a}}$ for unobserved items $i \in \mathcal{I}^-$:

$$\hat{\mu} = \frac{1}{|\mathcal{I}|}\Big(\sum_{i\in\mathcal{I}^+} s_i + \sum_{i\in\mathcal{I}^-}\boldsymbol{v}^T(\boldsymbol{x}_i)\hat{\boldsymbol{a}}\Big). \tag{42}$$

To estimate α^*, we maximize the likelihood

$$p_\alpha(\{s_i\}_{i\in\mathcal{I}}) = \prod_{i\in\mathcal{I}^+} p_0\big(1+\alpha(s_i-\hat{\mu})\big)\prod_{i\in\mathcal{I}^-}\Big(1-p_0\big(1+\alpha(s_i-\hat{\mu})\big)\Big), \tag{43}$$

w.r.t. α. However, $\{s_i\}_{i\in\mathcal{I}^-}$ are not observed. So, we plug in $s_i \approx \boldsymbol{v}^T(\boldsymbol{x}_i)\hat{\boldsymbol{a}}$ for $i \in \mathcal{I}^-$ and find

$$\hat{\alpha} = \arg\max_{\alpha}\; p_\alpha(\{s_i\}_{i\in\mathcal{I}^+}, \{\boldsymbol{v}^T(\boldsymbol{x}_i)\hat{\boldsymbol{a}}\}_{i\in\mathcal{I}^-}). \tag{44}$$

This maximization can be solved efficiently. In fact, p_α is a polynomial in α with $|\mathcal{I}|$ roots. Let $\mathcal{R}^+ = \{\frac{1}{\hat{\mu}-s_i}\}_{i\in\mathcal{I}^+}$ be the roots corresponding to $\mathcal{I}^+$ terms and $\mathcal{R}^- = \{\frac{1-p_0}{p_0(s_i-\hat{\mu})}\}_{\mathcal{I}^-}$ be the roots corresponding to $\mathcal{I}^-$ terms. These two sets together form all of the roots $\mathcal{R} = \mathcal{R}^+ \cup \mathcal{R}^-$ of p_α.We are only interested in the feasible region where $p_0\big(1+\alpha(s_i-\hat{\mu})\big)$ and $1-p_0\big(1+\alpha(s_i-\hat{\mu})\big)$ are positive for all i. Thus, for $r \in \mathcal{R}^+$, if $r > 0$, we are interested in $\alpha < r$, and if $r < 0$, we only need to search $\alpha > r$. One can see that similar conditions hold for $r \in \mathcal{R}^-$. Overall, our search space is limited to

$$\mathcal{A} = (\max_{r\in\mathcal{R}}\{r \mid r < 0\}, \min_{r\in\mathcal{R}}\{r \mid r > 0\}). \tag{45}$$

In other words, we need to search only the range between the two roots closest to the origin from left and right. Since p_α is a real-rooted polynomial and no other root is in this interval, p_α has a single mode in $\mathcal{A}$ and we can use simple algorithms like golden section search to find its maximizer.

4.2.3. Bias correction

Note that the expected value of the noise term in Equation 40 follows

$$\mathbb{E}\Big[(\hat{\boldsymbol{K}}+\boldsymbol{C})^{-1}\sum_{i\in\mathcal{I}^+}\boldsymbol{v}(\boldsymbol{x}_i)n_i\Big] = (\hat{\boldsymbol{K}}+\boldsymbol{C})^{-1}\sum_{i\in\mathcal{I}}\boldsymbol{v}(\boldsymbol{x}_i)\mathbb{E}[n_iO_i] \tag{46}$$

Now, we can leverage the estimated missing data model to find $\mathbb{E}[n_iO_i]$:

$$\begin{aligned}\mathbb{E}[n_iO_i] &= \mathbb{E}\big[n_i\mathbb{E}[O_i|n_i]\big] \\ &= \mathbb{E}\big[n_i(1+\hat{\alpha}(s_i-\hat{\mu}))\big] & (47)\\ &= \mathbb{E}\big[n_i(1+\hat{\alpha}(\boldsymbol{v}(\boldsymbol{x}_i)^T\boldsymbol{a}^*+n_i-\hat{\mu}))\big] = \hat{\alpha}\sigma_n^2. & (48)\end{aligned}$$

The last missing piece is σ_n^2. We can approximate it by $\hat{\sigma}_n^2 = \frac{1}{|\mathcal{I}^+|}\sum_{i\in\mathcal{I}^+}(s_i - \boldsymbol{v}^T(\boldsymbol{x}_i)\hat{\boldsymbol{a}})^2$. Putting these all together, we can subtract the expected noise term from $\hat{\boldsymbol{a}}$ and obtain a bias-corrected estimation:

$$\hat{\boldsymbol{a}}_{\mathrm{bc}} = \hat{\boldsymbol{a}} - \hat{\alpha}\hat{\sigma}_n^2(\hat{\boldsymbol{K}}+\boldsymbol{C})^{-1}\big(\sum_{i\in\mathcal{I}}\boldsymbol{v}(\boldsymbol{x}_i)\big) \tag{49}$$

It should be noted that all the parameters in Equation 49 are estimated from the observed data. In practice, we minimize the empirical loss and obtain $\hat{\boldsymbol{a}}$. Then, at the test time, we use Equation 49 to correct for the induced bias.

4.3. Frequency-aware regularization

An advantage of our framework is the possibility to have frequency domain interpretation from variables. For example, we might expect a user's general interest in romance movies to appear in lower frequencies, while his/her specific interest in contemporary vs. historical romance to exist in higher frequencies. Since user behavior is smooth, it is often easier to recover strong lower-frequency coefficients than higher-frequency ones. Therefore, we penalize higher-frequency coefficients with a larger regularization term. This can be seen as an indirect way of low-pass filtering.

Let $\boldsymbol{a}$ be a user or a cluster Fourier coefficient vector. Remember $\boldsymbol{a}$ is the vectorized form of the d-dimensional tensor A by following a lexicographic order (Equation 13). Therefore, we can rewrite the L_2 regularization of $\boldsymbol{a}$ as

$$\lambda\|\boldsymbol{a}\|^2 = \sum_{m_1=0}^{M}\cdots\sum_{m_d=0}^{M}\lambda \mathsf{A}_{m_1,\cdots,m_d}^2. \tag{50}$$

Table 1: Summary of the datasets.

Dataset	#User	#Item	#Rating	Density	Range
Jester 1	48,483	100	3,519,324	0.73	[-10, 10]
Synthetic	50	200	variable	variable	1, 2, ..., 5
ML-100k	943	1,682	100,000	0.063	1, 2, ..., 5
ML-1M	6,040	3,706	1,000,209	0.045	1, 2, ..., 5
Yahoo! Music	15,400	1000	311,704	0.020	1, 2, ..., 5
Coat	290	300	6,960	0.08	1, 2, ..., 5

In frequency-aware regularization, instead of equally weighting $\mathsf{A}_{m_1,\cdots,m_d}$s by a factor of λ, we propose to use a frequency-aware weight of $w(m_1, \cdots, m_d)$. The optimal choice of $w(\cdot)$ depends on our assumption about the signal bandwidth, noise model, and optimality criteria. As a proof of concept, we shall use a simple weighting that exponentially penalizes higher-frequency terms:

$$w(m_1, \cdots, m_d) = \lambda\gamma^{(\sum_{q=1}^{d} m_q)}. \tag{51}$$

Here, $\gamma \geq 1$ is a hyper-parameter that can be tuned by cross-validation.

5. Experiments

We consider six datasets for evaluating the performance of our proposed methods; they include the MovieLens-100k (ML-100k) and MovieLens-1M (ML-1M) datasets [10], the Yahoo! Music (R3) dataset[9], the Jester 1 dataset [8], the coat shopping dataset [23], and a synthetic dataset. The details of each are provided in Table 1. Next, we briefly introduce the datasets and the experiments. Details for each experiment and the results will follow in subsequent sections.

As explained in Section 2.2.3, our proposed scoring functions could be adapted to asymmetric data, where the number of users is much higher than the number of items (or vice versa). We use Jester 1 as a representative of asymmetric datasets to justify this argument in Section 5.1.

The synthetic data is created to assess our clustering methods; for this purpose, a number of cluster representations (Fourier coefficients) are chosen

[9]Yahoo! Webscope dataset (ydata-ymusic-rating-study-v1_0) available at `http://research.yahoo.com/Academic_Relations`

at random. Then, each user is placed randomly around one of the cluster representations. We study the performance of different clustering methods for different levels of clustering difficulty. See Section 5.2 for a detailed discussion.

We use the MovieLens datasets[10] ML-100k and ML-1M as two of the most common benchmarks for collaborative filtering to compare the error of our predicted ratings with well-established latent feature methods. We also assess our approach on the sparser Yahoo! Music dataset. Performance details are provided in Section 5.3.

Finally, the coat shopping dataset consisting of a missed-not-at-random training set and a missed-completely-at-random test set is used to evaluate the proposed debiasing methods. Check Section 5.4 for the details of the experiment and the results.

5.1. Adapting to asymmetric data

As we explained in Section 2.2.3, the more ratings we have per item or user, the more accurate is the estimated model for the item or the user. However, as the MF technique forces the users and items to be in the same vector space, the complexity of item/user representations is essentially equal. In contrast, we have control over the complexity of the functions in our approach by tuning their bandwidth M, while preserving the dimension of the space d (dimension of item representation).

To demonstrate the effect of changing M, we use the Jester 1 dataset, which has a large number of ratings per item. Data is split into 80%, 10%, and 10% partitions, for training, validation, and test, respectively. We did cross-validation to choose the appropriate regularization coefficient λ.

In this experiment, we used an item-based method where items are associated with smooth scoring functions (see Section 3.3). For each user, we used the alternating optimization (Section 3.1) without clustering to obtain user representations and item Fourier coefficient vectors.[11] Although the clustering method is helpful here, we did not cluster users/items to have a fair comparison with PMF [18] as the baseline. To train the PMF, we used the alternating least squares method. We kept $d = 2$ (the dimension of the user space) for all methods. In practice, nevertheless, PMF and our method

[10]`https://grouplens.org/datasets/movielens/`

[11]In the item-based setting, we represent users with d-dimensional vectors and find the item scoring functions.

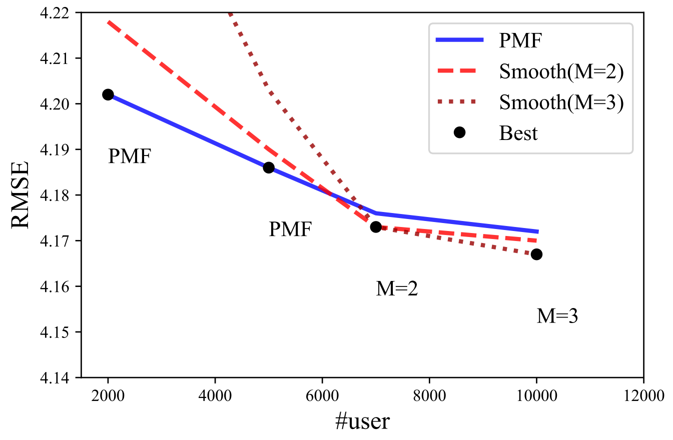


Figure 4: Keeping the dimension of user latent space d fixed, the RMSE of predicted ratings on the test set of Jester 1 dataset is plotted for PMF and our item-based method (for $M = 2$ and $M = 3$). The black dots mark the best performance among the three methods for each number of users.

use higher dimensional representations; the choice of $d = 2$ is solely for the purpose of insightful demonstrations.

Figure 4 shows the performance for $M = 2$ and $M = 3$ compared to the PMF baseline. The performance is reported in terms of the RMSE on the test data, and the best performance is marked. Two observations can be made from these numbers: First, the larger the number of users, the better the results for larger M. Second, by adaptive controlling of M, the performance of our method does not saturate and its gap with the baseline increases.

5.2. Evaluating k-representation clustering and interpretability of SmoothRecNet's soft clustering

In Section 3.1.2, we proposed the k-representation clustering and its boosted version. We also interpreted one of the SmoothRecNet's layers as a soft clustering mechanism. In this section, we investigate these clustering methods through experiments on synthetic data. Note that these methods do not explicitly penalize miss-clustering. Instead, they minimize the within-cluster reconstruction cost. When clusters are clearly distinguishable, we anticipate these methods to perform fairly well and show this in the following.

We use synthetic data to assess clustering methods. We consider 200 items in a 2D space and draw item representations independently from $\text{Uniform}(0,1)$. We assume there are 4 clusters and the [cluster] scoring functions have a bandwidth of $M = 3$. We draw 16D cluster Fourier coefficient vectors, also called cluster representations, from $\mathcal{N}(0, 0.1^2)$. Then, we randomly assign 50 users to the clusters. Users are placed within each cluster randomly around the cluster's representation in the frequency domain. More precisely, in cluster k with representation $\boldsymbol{a}_k$, we draw the user representation from $\boldsymbol{a}_k + \mathcal{N}(0, \sigma^2_{\text{within}})$. The within-cluster variance σ^2_{within} is a parameter that we tune to control the distinguishability of the clusters. With given user and item representations, we derive ground-truth ratings. We then, randomly include a subset of these ratings as observed ratings and control the density of observed ratings in our experiments.

To measure the matching between the identified clusters and the original ones, we employ the *adjusted rand index* (ARI) [32]. For two clusterings with the same domain $\mathcal{U}$, ARI can be calculated from their contingency matrix and intuitively shows the rate of agreement between the two clusterings if u is randomly chosen from $\mathcal{U}$[12]. ARI takes the maximum value of 1 for identical clusterings and the minimum value of 0 when the clusterings are perceived as fully random with respect to each other. ARI has the advantage of comparing two clusterings even with different numbers of clusters.

To use the ARI metric, we need a hard clustering of users. However, this is not the case in SmoothRecNet. Therefore, we associate each user in our user-layer in Figure 3 to the neuron (cluster) in the last hidden unit with the largest absolute weight. Although this technique violates the main goal of soft clustering, it provides us with a measure of clustering accuracy.

In the first experiment, we assume that all the clustering methods choose the correct number of clusters consistent with the true underlying data. For a density of observed data of 0.1, we vary σ^2_{within} and compare the ARI of the clustering methods. We measure the distinguishability of the clusters by the discrimination index, defined as the ratio of between-cluster to within-cluster standard deviations: $0.1/\sigma_{\text{within}}$. Figure 5a presents the ARI curves in terms of the discrimination index of the clusters. We observe that the boosted k-representation performs well when clusters have high discrimination indices. But its performance degrades when the clusters are cluttered.

[12]We use the reference implementation from `sklearn` python package.

Also, as expected, we obtain inferior results from the soft-clustering layer; however, since the ARI is meaningfully above zero, our interpretation of soft clustering is justified.

In the second experiment, for a fixed discrimination index of 10, we vary the density of observations and compare the methods in terms of their ARI. Figure 5b, shows ARI versus the density of the rating matrix. While the k-representation has the best performance at low densities, its performance drops when the density exceeds a threshold. This might be due to the involved regularization term.

Finally, in Figure 5c, we study the robustness to the error in estimating the number of clusters. Here, the true number of clusters is kept fixed at four and the algorithm's number of clusters is varied. One can see that k-representation and its boosted version are fairly robust and none of them dominates the other one in all regions.

5.3. Evaluating the predicted ratings

5.3.1. Data split

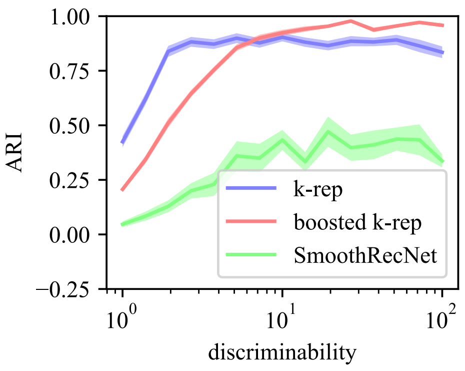
For ML-100k, we adhere to the prescribed test sets, each comprising of 20% of the data. We use the 80% data allocated for the training, we use 75% for training and 5% for validation. The validation data is employed for fine-tuning the hyperparameters. The data split is consistent across all the methods compared in this section. We report the median root mean-squared error (RMSE) obtained from each of the five test sets as the final performance of the method on ML-100k.

For ML-1M and Yahoo! Music, 85%, 5%, and 10% of the data are used for training, validation, and test, respectively. These numbers are consistent across all methods.

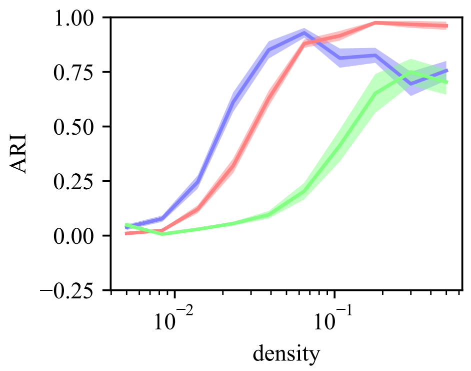
5.3.2. Model parameters

Proposed methods. The key model parameters used for each of the proposed methods (alternating optimization and SmoothRecNet) and each dataset are provided in Table 2. Other parameters can be found in our publicly available python package[13]. For the alternating optimization, we experimented with values of d ranging from 2 to 7 and M from 3 to 7. Additionally, we tested with 10 and 15 weak learners for boosted k-representation. Regarding

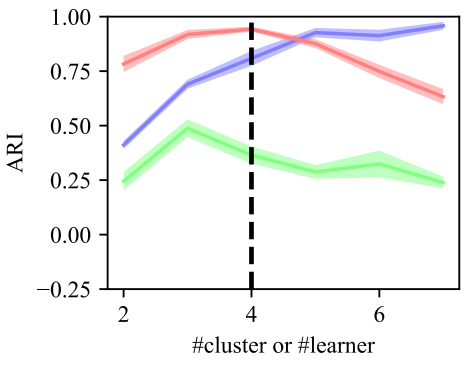
[13]`https://github.com/alishiraliGit/collaborative-filtering-in-the-frequency-domain`



(a)



(b)



(c)

Figure 5: Evaluation of the clustering methods on synthetic data.

the L_2 regularization parameter, we considered values of 1, 3, 10, and 30. For SmoothRecNet, we explored similar options for d and M. We limited our exploration to just two configurations for the number of hidden layers, corresponding to either 5 or 10 soft clusters. Training SmoothRecNet on our largest benchmark, ML-1M, using a single V100 GPU takes less than 30 minutes and requires less than 1 GB of GPU RAM.

Baseline methods.. We used PMF [18], BPMF [21], NNMF [7], MLP, and NCF [11] as the baselines. These are all well-known and widely adopted latent feature methods that perform closely to the state-of-the-art collaborative filtering methods or even outperform many of the recent methods if tuned extesnively [20]. Note that MLP and NCF were originally introduced

Table 2: Model parameters for the proposed methods. Numbers in parentheses show the different values used for the item-based setting (if any). For SmoothRecNet, we used a dropout ratio of 0.1 in all settings.

Method	**Dataset**	d	M	**Other parameters**
Alt. optim.	ML-100k	3	7	boosted k-rep. w/ 10 learners, $\lambda = 30$
	ML-1M	3	7	boosted k-rep. w/ 15 learners, $\lambda = 1$
	Yahoo!	2	3 (4)	boosted k-rep. w/ 10 learners, $\lambda = 10$
SmoothRecNet	ML-100k	5	3	$n_1 = n_3 = 5\,(10), n_3 = 50\,(100)$
	ML-1M	4	5	$n_1 = n_3 = 10, n_2 = 100$
	Yahoo!	7 (3)	3 (5)	$n_1 = n_3 = 5, n_2 = 50$

for implicit feedback prediction but we modified and optimized them for general rating prediction tasks. For PMF, we tuned the dimension of the latent feature space as 10, 20, and 100[14]. For all three datasets, we found that a 10-dimensional space with 0.01 regularization was the best parameter setting. For BPMF, we used reference parameters, employing a 10-dimensional latent feature space and setting β to 5 for the prior on the observation noise. We ran the algorithm for 50 epochs. For MLP and NCF, we conducted tuning experiments for the embedding size, considering values of 16, 32, 64, and 128. In the case of NCF, we additionally tuned the MF embedding size to be 4, 8, 16, and 32. For MLP, we set the embedding size to 16, 32, and 64 on ML-100k, ML-1M, and Yahoo! Music, respectively. For NCF, we found that the embedding size of 32 was the optimal choice for ML-100k and ML-1M, while the embedding size of 64 was ideal for Yahoo! Music. Additionally, we found that the MF embedding size of 8 yielded the best results for NCF.

5.3.3. Performance comparison in rating prediction

We compare the RMSE of prediction on test data of ML-100k, ML-1M, and Yahoo! Music in Table 3. The performance of our proposed methods is evaluated after linearly combining user-based and item-based predictors as explained in Section 3.3.

One can see that SmoothRecNet, despite its simplicity and interpretability, outperforms other methods on ML-100k and Yahoo! Music, and closely

[14]We used a reference implementation for PMF and BPMF from `https://www.cs.toronto.edu/~rsalakhu/BPMF.html`

Table 3: RMSE comparison on ML-100k, ML-1M, and Yahoo! Music.

Method	**ML-100k**	**ML-1M**	**Yahoo! Music**
PMF	0.951	0.877	1.274
BPMF	0.920	<u>0.849</u>	1.296
NNMF	0.948	0.892	1.280
MLP	0.926	0.881	1.224
NCF	0.920	0.870	1.221
Alternating optimization (ours)	0.920	0.865	<u>1.215</u>
SmoothRecNet (ours)	<u>0.914</u>	0.858	<u>1.215</u>

follows BPMF on ML-1M. This comparison is insightful as NNMF, MLP, and NCF are all neural network extensions to MF. However, we believe that these extensions might not be consistent with the smoothness of human behavior which is the basis for the development of our algorithms.

We should mention that our comparison is not comprehensive here. Specifically, graph feature models are excluded from our analysis; for example, graph convolutional matrix completion (GC-MC) [1] outperforms our method on ML-100k and ML-1M datasets. We limited our scope to the state-of-the-art latent feature models which have similar complexity to our method. As discussed earlier, latent feature models are easy to scale and interpret, and their performance can be improved by applying standard techniques to augment them with graph features. Moreover, recent investigations have shown that by extensive fine-tuning of the baseline latent feature models, one can surpass nearly all recent methods, making matrix factorization techniques a challenging baseline to surpass [20].

We would also like to emphasize that our evaluation of predicted rating errors is intended to demonstrate sufficient predictive capability rather than competitiveness. In fact, progress in predictive performance has been relatively slow in recent years, with other aspects of recommender systems taking center stage. These include the capacity to offer interpretable recommendations and addressing potential distribution shifts from training to testing data. Our frequency-domain representations offer an interpretable and straightforward formulation for recommendation that can tackle some of these challenges from a fresh perspective.

5.4. Robustness to missing-not-at-random

The coat shopping dataset [23] is special in the way that it provides both missed-not-at-random (MNAR) and missed-completely-at-random (MCAR) data. To collect this dataset, annotators were first asked to explore an online store and shop coats. Then, they were asked to rate 24 coats they explored (self-selected and so MNAR) and 16 randomly picked coats (MCAR).

We use the MNAR part of the data to train our algorithms (10% for validation) and use the MCAR part as the test data. For methods that use IPS, we use the propensity scores provided by dataset creators. These scores are obtained from a standard regularized logistic regression using side information (e.g., gender, coat type, etc.). As the baseline, we compare against MF corrected by IPS as it was shown to be a promising technique in the MNAR setting [23]. We use our (user-based) alternating optimization method with $d = 2$, $m = 3$, which uses boosted k-representation with $k = 10$ for clustering. In terms of complexity, our algorithm and MF are equally simple. We shall compare various versions of our alternating optimization method augmented with IPS, test-time bias correction, and frequency-aware regularization ($\gamma = 1.2$).

Table 4 shows RMSE on the MCAR portion of the data. First of all, as we previously observed, alternating optimization is superior to MF and this is preserved with IPS as well. Second, the standard methods of MF and our alternating optimization perform poorly when MNAR is not addressed. This, once again shows the significance of the distribution shift from observed to unobserved data in the offline evaluation of recommender systems. Third, unlike the methods with IPS, our bias correction does not require any explicit propensity score. It also does not require any retraining. Despite this simplicity, it has been extremely helpful in mitigating the effect of MNAR. Last but not least, frequency-aware regularization rooted in our intuition from the frequency domain behavior of the users, although not directly related to the MNAR effect, is helpful in reducing the effect of distribution shift. Overall, the flexibility of our formulation allows for various ways to address MNAR and they all turned out to be effective.

6. Conclusion

In this work, we considered the problem of collaborative filtering by modeling user-to-item scores as smooth functions. We first motivated the use of

Table 4: MSE on missed-completely-at-random test set of Coat.

Method	**Coat**
MF	1.202
MF with IPS	1.093
Alternating optimization	1.123
Alt. optim. w/ IPS	1.077
Alt. optim. w/ IPS and freq.-aware reg.	1.071
Alt. optim. w/ bias correction	1.070
Alt. optim. w/ bias correction and freq.-aware reg.	<u>1.067</u>

such a class of functions and developed the mathematical background to formulate the problem in this manner. Next, we proposed practical algorithms to learn user and item representations. The algorithms were capable of implicitly or explicitly clustering the users while learning the representations. In accordance with previous findings, we showed that this clustering is helpful in having better predictions.

On the empirical side, we tested our methods on 6 different datasets. First, we showed our method's capability of handling asymmetric data where there are many more users than items (or vice versa). Second, using a synthetic dataset, we showed how the introduced clustering methods work in practice. We further showed that weights in the proposed neural network architecture could be attributed to the soft clustering of the users. Third, we used three benchmark datasets to compare the predictive performance of the proposed methods with widely adopted latent feature models of similar complexity. Despite the simplicity and interpretability of the proposed methods, we achieved comparable performance with the best-performing latent feature models.

Last but not least, we extended our framework to address missed-not-at-random settings, where the distribution of unobserved ratings possibly deviates from the observed ones. We showed how the popular technique of inverse propensity scoring can be incorporated into our method. Powered by the simplicity of our formulation, we introduced test-time bias correction that can effectively cancel out the missed-not-at-random effect. We also proposed frequency-aware regularization as a proof of concept of how frequency domain interpretation of the representations can be helpful in making learning more robust. We demonstrated the effectiveness of all the aforementioned methods on a dataset that unconventionally had provided a portion of data missed completely at random.

In sum, the proposed frequency domain representation provides a natural and flexible way to model users and items. Beyond predictive power, this space can provide a new lens to the common issues of recommender systems. For example, the popularity bias can be formalized as the dominance of lower-frequency terms or the level of personalization can be controlled by the user based on a single bandwidth parameter. We encourage future works pursuing these new formalisms and possibly new solutions in the frequency domain.

References

[1] Berg, R.v.d., Kipf, T.N., Welling, M., 2017. Graph convolutional matrix completion. arXiv preprint arXiv:1706.02263 .

[2] Byrd, R.H., Lu, P., Nocedal, J., Zhu, C., 1995. A limited memory algorithm for bound constrained optimization. SIAM Journal on scientific computing 16, 1190–1208.

[3] Cheng, H.T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M., et al., 2016. Wide & deep learning for recommender systems, in: Proceedings of the 1st workshop on deep learning for recommender systems, pp. 7–10.

[4] Davidson, J., Liebald, B., Liu, J., Nandy, P., Van Vleet, T., Gargi, U., Gupta, S., He, Y., Lambert, M., Livingston, B., et al., 2010. The youtube video recommendation system, in: Proceedings of the fourth ACM conference on Recommender systems, pp. 293–296.

[5] Dudík, M., Langford, J., Li, L., 2011. Doubly robust policy evaluation and learning. arXiv preprint arXiv:1103.4601 .

[6] Dwork, C., Feldman, V., Hardt, M., Pitassi, T., Reingold, O., Roth, A.L., 2015. Preserving statistical validity in adaptive data analysis, in: Proceedings of the forty-seventh annual ACM symposium on Theory of computing, pp. 117–126.

[7] Dziugaite, G.K., Roy, D.M., 2015. Neural network matrix factorization. arXiv preprint arXiv:1511.06443 .

[8] Goldberg, K., Roeder, T., Gupta, D., Perkins, C., 2001. Eigentaste: A constant time collaborative filtering algorithm. information retrieval 4, 133–151.

[9] Gomez-Uribe, C.A., Hunt, N., 2015. The netflix recommender system: Algorithms, business value, and innovation. ACM Transactions on Management Information Systems (TMIS) 6, 1–19.

[10] Harper, F.M., Konstan, J.A., 2015. The movielens datasets: History and context. Acm transactions on interactive intelligent systems (tiis) 5, 1–19.

[11] He, X., Liao, L., Zhang, H., Nie, L., Hu, X., Chua, T.S., 2017. Neural collaborative filtering, in: Proceedings of the 26th international conference on world wide web, pp. 173–182.

[12] Hernández-Lobato, J.M., Houlsby, N., Ghahramani, Z., 2014. Probabilistic matrix factorization with non-random missing data, in: International Conference on Machine Learning, PMLR. pp. 1512–1520.

[13] Huang, W., Marques, A.G., Ribeiro, A., 2017. Collaborative filtering via graph signal processing, in: 2017 25th European Signal Processing Conference (EUSIPCO), IEEE. pp. 1094–1098.

[14] Koren, Y., Bell, R., Volinsky, C., 2009. Matrix factorization techniques for recommender systems. Computer 42, 30–37.

[15] Lee, J., Kim, S., Lebanon, G., Singer, Y., Bengio, S., 2016. Llorma: Local low-rank matrix approximation .

[16] Little, R.J., Rubin, D.B., 2019. Statistical analysis with missing data. volume 793. John Wiley & Sons.

[17] Mao, K., Zhu, J., Xiao, X., Lu, B., Wang, Z., He, X., 2021. Ultragcn: ultra simplification of graph convolutional networks for recommendation, in: Proceedings of the 30th ACM International Conference on Information & Knowledge Management, pp. 1253–1262.

[18] Mnih, A., Salakhutdinov, R.R., 2008. Probabilistic matrix factorization, in: Advances in neural information processing systems, pp. 1257–1264.

[19] Pradel, B., Usunier, N., Gallinari, P., 2012. Ranking with non-random missing ratings: influence of popularity and positivity on evaluation metrics, in: Proceedings of the sixth ACM conference on Recommender systems, pp. 147–154.

[20] Rendle, S., Zhang, L., Koren, Y., 2019. On the difficulty of evaluating baselines: A study on recommender systems. arXiv preprint arXiv:1905.01395 .

[21] Salakhutdinov, R., Mnih, A., 2008. Bayesian probabilistic matrix factorization using markov chain monte carlo, in: Proceedings of the 25th international conference on Machine learning, pp. 880–887.

[22] Schnabel, T., Swaminathan, A., Singh, A., Chandak, N., Joachims, T., 2016a. Recommendations as treatments: Debiasing learning and evaluation, in: international conference on machine learning, PMLR. pp. 1670–1679.

[23] Schnabel, T., Swaminathan, A., Singh, A., Chandak, N., Joachims, T., 2016b. Recommendations as treatments: Debiasing learning and evaluation, in: international conference on machine learning, PMLR. pp. 1670–1679.

[24] Sedhain, S., Menon, A.K., Sanner, S., Xie, L., 2015. Autorec: Autoencoders meet collaborative filtering, in: Proceedings of the 24th international conference on World Wide Web, pp. 111–112.

[25] Shi, S., Zhang, M., Liu, Y., Ma, S., 2018. Attention-based adaptive model to unify warm and cold starts recommendation, in: Proceedings of the 27th ACM international conference on information and knowledge management, pp. 127–136.

[26] Shirali, A., 2022. Sequential nature of recommender systems disrupts the evaluation process, in: Advances in Bias and Fairness in Information Retrieval: Third International Workshop, BIAS 2022, Stavanger, Norway, April 10, 2022, Revised Selected Papers, Springer. pp. 21–34.

[27] Strehl, A., Langford, J., Li, L., Kakade, S.M., 2010. Learning from logged implicit exploration data. Advances in neural information processing systems 23.

[28] Strub, F., Gaudel, R., Mary, J., 2016. Hybrid recommender system based on autoencoders, in: Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, pp. 11–16.

[29] Swaminathan, A., Joachims, T., 2015. The self-normalized estimator for counterfactual learning. advances in neural information processing systems 28.

[30] Tran, C., Kim, J.Y., Shin, W.Y., Kim, S.W., 2019. Clustering-based collaborative filtering using an incentivized/penalized user model. IEEE Access 7, 62115–62125.

[31] Ungar, L.H., Foster, D.P., 1998. Clustering methods for collaborative filtering, in: AAAI workshop on recommendation systems, Menlo Park, CA. pp. 114–129.

[32] Vinh, N.X., Epps, J., Bailey, J., 2010. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. The Journal of Machine Learning Research 11, 2837–2854.

[33] Wang, X., He, X., Wang, M., Feng, F., Chua, T.S., 2019a. Neural graph collaborative filtering, in: Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval, pp. 165–174.

[34] Wang, X., Zhang, R., Sun, Y., Qi, J., 2019b. Doubly robust joint learning for recommendation on data missing not at random, in: International Conference on Machine Learning, PMLR. pp. 6638–6647.

[35] Wu, J., Wang, X., Feng, F., He, X., Chen, L., Lian, J., Xie, X., 2021. Self-supervised graph learning for recommendation, in: Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval, pp. 726–735.

[36] Xia, L., Huang, C., Xu, Y., Zhao, J., Yin, D., Huang, J., 2022. Hypergraph contrastive collaborative filtering, in: Proceedings of the 45th

International ACM SIGIR conference on research and development in information retrieval, pp. 70–79.

[37] Yu, X., Chu, Y., Jiang, F., Guo, Y., Gong, D., 2018. Svms classification based two-side cross domain collaborative filtering by inferring intrinsic user and item features. Knowledge-Based Systems 141, 80–91.

[38] Yu, X., Jiang, F., Du, J., Gong, D., 2019. A cross-domain collaborative filtering algorithm with expanding user and item features via the latent factor space of auxiliary domains. Pattern Recognition 94, 96–109.

[39] Yu, X., Peng, Q., Xu, L., Jiang, F., Du, J., Gong, D., 2021. A selective ensemble learning based two-sided cross-domain collaborative filtering algorithm. Information Processing & Management 58, 102691.

[40] Zhang, S., Yao, L., Sun, A., Tay, Y., 2019. Deep learning based recommender system: A survey and new perspectives. ACM Computing Surveys (CSUR) 52, 1–38.

[41] Zhang, S., Yao, L., Sun, A., Wang, S., Long, G., Dong, M., 2018. Neurec: On nonlinear transformation for personalized ranking. arXiv preprint arXiv:1805.03002 .

[42] Zheng, L., Lu, C.T., Jiang, F., Zhang, J., Yu, P.S., 2018. Spectral collaborative filtering, in: Proceedings of the 12th ACM conference on recommender systems, pp. 311–319.

[43] Zhu, C., Byrd, R.H., Lu, P., Nocedal, J., 1997. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. ACM Transactions on Mathematical Software (TOMS) 23, 550–560.