Optimizing Factorization Machines for Top-N Context-aware Recommendations

Fajie Yuan[1], Guibing Guo[2], Joemon M. Jose[1], Long Chen[1], Haitao Yu[3], and Weinan Zhang[4]

[1]University of Glasgow, UK
f.yuan.1@research.gla.ac.uk,{Joemon.Jose,Long.Chen}@glasgow.ac.uk
[2]Northeastern University, China [3]University of Tsukuba, Japan [4]Shanghai Jiao Tong University, China
guogb@swc.neu.edu.cn, yuhaitao@slis.tsukuba.ac.jp, wnzhang@sjtu.edu.cn

Abstract. Context-aware Collaborative Filtering (CF) techniques such as Factorization Machines (FM) have been proven to yield high precision for rating prediction. However, the goal of recommender systems is often referred to as a top-N item recommendation task, and item ranking is a better formulation for the recommendation problem. In this paper, we present two collaborative rankers, namely, Ranking Factorization Machines (RankingFM) and Lambda Factorization Machines (LambdaFM), which optimize the FM model for the item recommendation task. Specifically, instead of fitting the preference of individual items, we first propose a RankingFM algorithm that applies the cross-entropy loss function to the FM model to estimate the pairwise preference between individual item pairs. Second, by considering the ranking bias in the item recommendation task, we design two effective *lambda*-motivated learning schemes for RankingFM to optimize desired ranking metrics, referred to as LambdaFM. The two models we propose can work with any types of context, and are capable of estimating latent interactions between the context features under sparsity. Experimental results show its superiority over several state-of-the-art methods on three public CF datasets in terms of two standard ranking metrics.

Keywords: Context-aware; Learning to rank; Factorization Machines; RankingFM; LambdaFM

1 Introduction

Commonly used recommendation techniques such as collaborative filtering (CF) have gained much attention in recent years. However, typical collaborative filtering (CF) methods mainly focus on mining interactions between users and items without considering the additional context which the users or items are associated with [21]. For example, in a music recommender system, the location of the user and the time of the user-item interaction may be important contextual factors when the user listened to a song. Ignoring the contextual information may result in considerable degradation in recommendation performance. Currently,

there are some hybrid approaches performing pre- or post-filtering of the input data to make standard methods context-aware. Although such ad-hoc strategies may work in practice, they suffer from two drawbacks [10, 21]: (1) pre- or post-filtering the data based on the context can potentially lead to information loss about the interactions between different contextual variables; (2) all steps in the process need supervision and manual tuning. On the other hand, a variety of specialized models designed for specific tasks, such as TimeSVD [11] and Tensor Factorization [18], are able to leverage contextual information, but they rely on very strict assumptions, which make them cumbersome to incorporate different types of context and usually require complicated inference algorithms. Therefore, the models capable of integrating any types of context are more practical, as well as more elegant in theory. So far, two of the most flexible and effective methods for context modelling are Multiverse Recommendation [10] and Factorization Machines (FM) [17]. Unfortunately, Multiverse Recommendation relies on Tucker decomposition, which leads to $O(k^m)$ computational complexity, where k is the dimensionality of factorization and m is the number of predictor variables involved [21]. In contrast, FM enjoys linear complexity (both in k and m), which gives fast learning and prediction with contextual features.

It has been recognized that both Multiverse Recommendation and FM were originally designed for the rating prediction task based on explicit user feedback [6, 21]. However, it is a commonplace that in real-world scenarios most observed feedback is not explicit but implicit [20]. Typical implicit feedback includes the number of purchases, clicks, played songs, etc., and thus it is much more accessible, because the user does not have to express his feelings explicitly [19]. As a result, implicit feedback is often one-class, i.e., only positive class is available. In addition, for item recommendation task, the recommendation accuracy near the top of the ranked list is usually more important than that at the end of the list, known as the top-N (item) ranking task. Some recent work has shown that rating prediction algorithms optimized for error metrics such as RMSE (root mean squared error) empirically do not guarantee accuracy in terms of top-N item recommendations [6].

To address the above drawbacks, we propose to optimize FM for the item recommendation task based on implicit feedback, which is also knowns as One-Class Collaborative Filtering (OCCF). More specifically: Firstly, we present RankingFM, which adopts FM as a ranking function to model the interactions between context features, and apply it to the Learning-to-Rank (LtR) approach by using pairwise cross-entropy (CE) loss. We propose to optimize the RankingFM by widely used stochastic gradient descent method. Secondly, inspired by LambdaRank [15], we explore to further improve the top-N recommendation performance of RankingFM by adapting the original lambda weighting function with two alternative sampling schemes, referred to as LambdaFM[1]. Lastly, we carry out a set of experiments on three public datasets. The results indicate that our proposed methods (i.e., RankingFM and LambdaFM) achieve superior recommendation quality in terms of two standard ranking metrics. In particular,

[1] A full version of LambdaFM has been published at CIKM'16 [26].

LambdaFM largely outperforms a bunch of strong baselines for top-N recommendations.

2 Related Work

Learning-to-Rank. Recently, Learning-to-Rank (LtR) has been attracting broad attention due to its effectiveness and importance in machine learning community. There are two major approaches, namely, pairwise [1, 18] and listwise approaches [3, 15]. Specifically, the pairwise ranking usually treats an objective pair as an 'instance' in learning. For example, Herbrich et al. [8] employed the approach and utilized the SVM technology to build a classifier, referred to as Ranking SVM; Burges et al. [1] adopted cross-entropy and gradient descent to train a Neural Network model, known as RankNet. Empirically, pairwise methods perform better than traditional pointwise methods. However, typical pairwise objective functions are devised to maximize the AUC metric, which is clearly position-independent. But for item recommendation, the recommendation quality is highly position-biased because the accuracy near the top of the ranked list is usually more important. In this regard, pairwise loss functions might still be a suboptimal scheme for the top-N item ranking task. In contrast, listwise approaches address the problem more directly because the models are usually formalized to optimize a specific ranking measure. Generally, it is difficult to directly optimize the ranking metrics because they are either flat or non-differentiable. One way to solve this problem is to propose smooth approximations of the target measures. For example, Shi et al. proposed smooth variants of MAP [24] and Mean Reciprocal Rank (MRR) [25] to optimize ranking performance. The other way is the lambda-based approach, such as LambdaRank[15] and LambdaMart [2], which is designed to add listwise information into pairwise implementation to bypass the major challenges of traditional listwise methods.

Factorization models. Recommender systems (RS) have two characteristics that distinguish themselves from conventional LtR (e.g., Ranking SVM, RankNet) in web search: (1) The user-item matrix is usually highly sparse e.g., $\geq 95\%$ in most scenarios where the conventional LtR is likely to fail [16, 21]; (2) RS aim at personalization, which means each user should receive one personalized ranking, whereas the conventional LtR learns only one ranking for a query, which, in effect, is non-personalization [20]. To tackle the above problems, researchers have proposed factorization models for recommendation tasks. Specifically, a series of matrix factorization (MF) based algorithms have been devised in the literature, e.g., Singular Value Decomposition (SVD) [11], Tensor Factorization (TF) [22], Probability Matrix Factorization (PMF) [23]. In particular, Rendle [16] unified factorization based models by developing a general predictor called Factorization Machines (FM), and showed that FM worked in linear time and can mimic several state-of-the-art MF models by feature engineering. Furthermore, FM demonstrates high recommendation accuracy for rating prediction by mining the latent interactions between pairwise features in sparse settings [16]. However, it has been pointed out that the least square based loss for rating prediction is suboptimal for item recommendation task [6, 18]. Accordingly, a variety

of ranking-based MF models have been proposed, e.g., WRMF [9] (pointwise), PITF [22] & RTF [18] (pairwise) and CLiMF [25] (listwise), which, however, were designed for specific tasks (e.g., tag recommendation) and cannot handle general scenarios of context-aware recommendations.

In our work, we adapt FM to RankingFM by applying the pairwise cross-entropy loss, and then explore to improve the way of pairwise learning by optimizing a rank biased performance measure. In contrast to the previous work, our proposed method is a general context-aware algorithm that is capable of effectively optimizing item ranking performance.

3 Ranking Factorization Machines

In this section, we first briefly review Factorization Machines (FM), and then elaborate our RankingFM algorithm. Lastly, the stochastic gradient descent (SGD) is applied to train the RankingFM model.

3.1 Factorization Machines

FM is a state-of-the-art pointwise prediction model, which is capable of capturing all nested interactions up to order d among n input variables in $\boldsymbol{x}$ with a factorized representation. For a detailed description, please refer to Rendle [17]. The FM model of order $d = 2$ is definied as:

$$\hat{y}(\boldsymbol{x}) = \underbrace{w_0 + \sum_{i=1}^{n} w_i x_i}_{\text{linear}} + \underbrace{\sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle x_i x_j}_{\text{polynomial}} \tag{1}$$

where the model parameters $\Theta = \{w_0, w_1, ..., w_n, v_{1,1}, ..., v_{n,k}\}$ to be estimated are: $w_0 \in \mathbb{R}, \boldsymbol{w} \in \mathbb{R}^n, \boldsymbol{V} \in \mathbb{R}^{n \times k}$, and $\langle \cdot, \cdot \rangle$ denotes the dot product of two vectors of size k:

$$\tau_{i,j} \approx \langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle = \sum_{f=1}^{k} v_{i,f} \cdot v_{j,f} \tag{2}$$

A row vector $\boldsymbol{v}_i$ of V is the i-th variable with k factors. The linear term of the FM model is identical to a linear regression model. The polynomial term models the interaction between the i-th and j-th variables by using a factorized parametrization $\langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle$ instead of an independent parameter $\tau_{i,j}$. In [16], it shows that FM can be computed in linear runtime $O(kn)$ because Eq. (1) can be reformulated as:

$$\hat{y}(\boldsymbol{x}) = w_0 + \sum_{i=1}^{n} w_i x_i + \frac{1}{2} \sum_{f=1}^{k} \Big(\big(\sum_{i=1}^{n} v_{i,f} x_i \big)^2 - \sum_{i=1}^{n} v_{i,f}^2 x_i^2 \Big) \tag{3}$$

3.2 RankingFM Framework

FM is recognized as being very successful for a variety of prediction problems with variables each of which may have interactions with one another. Specifically, pointwise error loss functions are adopted in the latent factor model for rating

prediction. However, as previously mentioned, the pointwise optimization results in a suboptimal solution for the item recommendation task, which is known as a ranking task. Thus we aim to extend FM to a Ranking FM approach by applying pairwise LtR techniques.

Consider that the learning algorithm is given a set of pairs of samples (a, b), with known probabilities $\overline{P}_{ab}$ that sample a will be ranked higher than sample b. Also, there exists an input vector $\boldsymbol{x} \in \mathbb{R}^n$, where n is the number of features. Meanwhile there is an output space of ranks represented by label $\boldsymbol{y} = \{y_1, y_2, ..., y_L\}$ with the number of ranks L. We denote the modeled posterior $P(\boldsymbol{x}^a \triangleright \boldsymbol{x}^b)$ by P_{ab}. Then $s_a = \hat{y}(\boldsymbol{x}^a)$ (i.e., Eq. (1)) and $s_{ab} = s_a - s_b = \hat{y}(\boldsymbol{x}^a) - \hat{y}(\boldsymbol{x}^b)$. Finally, the deviation between P_{ab} and $\overline{P}_{ab}$ can be formulated by cross-entropy (CE) loss [1]:

$$C_{ab} = C\left(s_{ab}\right) = -\overline{P}_{ab} \log P_{ab} - \left(1 - \overline{P}_{ab}\right) \log\left(1 - P_{ab}\right) \tag{4}$$

where the two outputs s_a, s_b of the models are mapped into a probability using sigmoid function, i.e., $P_{ab} = \frac{1}{1+e^{-(s_a - s_b)}}$.

In the case of Recommender Systems (RS)[2], for a given user $u \in \mathcal{U}$, let $S_{ab} \in \{0, \pm 1\}$ be defined as 1 if u prefers item a over item b, -1 if opposite, and 0 if u has the same preference of them. $\overline{P}_{ab}$ is assumed to be deterministically known from the ground truth, so that $\overline{P}_{ab} = \frac{1}{2}(1 + S_{ab})$. By combining the above equations, C_{ab} becomes:

$$C_{ab} = \frac{1}{2}\left(1 - S_{ab}\right)\left(s_a - s_b\right) + \log\left(1 + e^{-(s_a - s_b)}\right) \tag{5}$$

The difference of s_a and s_b can be computed with the computational complexity of $O(kn)$ by applying Eq. (3):

$$\begin{aligned} s_{ab} = \sum_{i=1}^{n} w_i (x_i^a - x_i^b) - \frac{1}{2} \sum_{f=1}^{k} \Big(\sum_{i=1}^{n} v_{i,f}^2 x_i^{a\,2} - \sum_{i=1}^{n} v_{i,f}^2 x_i^{b\,2} \Big) \\ + \frac{1}{2} \sum_{f=1}^{k} \Big(\Big(\sum_{i=1}^{n} v_{i,f} x_i^a \Big)^2 - \Big(\sum_{i=1}^{n} v_{i,f} x_i^b \Big)^2 \Big) \end{aligned} \tag{6}$$

The objective of CE is to train the scoring function (i.e., FM) so that the loss of ordering probability estimation can be minimized by:

$$C = \sum_{a,b \in D_s} C_{a,b} + \sum_{\theta \in \Theta} \gamma_\theta ||\theta||^2 \tag{7}$$

where D_s represents all the pair collections, $|| \cdot ||^2$ is the Frobernius norm and γ_θ is a hyper-parameter for the L2 regularization term.

3.3 Optimization Methods

We adopt the stochastic gradient descent (SGD) to optimize the loss function. By differentiating Eq. (17), the parameter θ can be updated:

$$\theta \leftarrow \theta - \eta \left(\frac{\partial C_{ab}}{\partial \theta} + \gamma_\theta \theta \right) \tag{8}$$

[2] User-item pairs function similarly as query-url pairs in the conventional LtR task.

Algorithm 1 RankingFM Learning

Input: Training dataset, regularization parameters γ, learning rate η
Output: $\Theta = (\boldsymbol{w}, \boldsymbol{V})$
Initialize Θ: $\boldsymbol{w} \leftarrow (0, ..., 0)$; $\boldsymbol{V} \sim \mathcal{N}(0, 0.1)$;
repeat
  for a, b with different labels given by u **do**
    $s_a = \hat{y}(\boldsymbol{x}^a)$, $s_b = \hat{y}(\boldsymbol{x}^b)$
    for $f \in \{1, ..., k\}$ **do**
      for $i \in \{1, ..., n\} \wedge x_i \neq 0$ **do**
        Update $v_{i,f}$ according to Eq. (13)
      end for
    end for
    for $i \in \{1, ..., n\} \wedge x_i \neq 0$ **do**
      Update w_i according to Eq. (14)
    end for
  end for
until convergence
return Θ

Algorithm 2 RankingFM Learning for OCCF

Uniformly draw u from $\mathcal{U}$
Uniformly draw a from $\mathcal{A}$
Uniformly draw b from $\mathcal{I} \backslash \mathcal{A}$

where

$$\frac{\partial C_{ab}}{\partial \theta} = \frac{\partial C_{ab}}{\partial s_a}\frac{\partial s_a}{\partial \theta} + \frac{\partial C_{ab}}{\partial s_b}\frac{\partial s_b}{\partial \theta} \tag{9}$$

$\frac{\partial C_{ab}}{\partial s_a}$ and $\frac{\partial C_{ab}}{\partial s_b}$ are the learning weights (i.e., the strength for updating θ), defined as:

$$\frac{\partial C_{ab}}{\partial s_a} = \left(\frac{1 - S_{ab}}{2} - \frac{1}{1 + e^{(s_a - s_b)}}\right) = -\frac{\partial C_{ab}}{\partial s_b} \tag{10}$$

According to Eq.(8)-(10), we obtain:

$$\theta \leftarrow \theta - \eta\left(\left(\frac{1 - S_{ab}}{2} - \frac{1}{1 + e^{(s_a - s_b)}}\right)\left(\frac{\partial (s_a - s_b)}{\partial \theta}\right) + \gamma_\theta \theta\right) \tag{11}$$

According to the property of Multilinearity [17], the gradient of FM can be derived:

$$\frac{\partial \hat{y}(\boldsymbol{x}^a)}{\partial \theta} = \begin{cases} 1 & \text{if } \theta \text{ is } w_0 \\ x_i^a & \text{if } \theta \text{ is } w_i \\ x_i^a \sum_{j=1}^{n} v_{j,f} x_j^a - v_{i,f} x_i^{a\,2} & \text{if } \theta \text{ is } v_{i,f} \end{cases} \tag{12}$$

By combining Eqs. (11)-(12), we have:

$$\begin{aligned} v_{i,f} \leftarrow v_{i,f} - \eta\Big(&\left(\frac{1 - S_{ab}}{2} - \frac{1}{1 + e^{(s_a - s_b)}}\right) \\ &\left(\sum_{j=1}^{n} v_{j,f}(x_i^a x_j^a - x_i^b x_j^b) - v_{i,f}(x_i^{a\,2} - x_i^{b\,2})\right) + \gamma_{v_{i,f}} v_{i,f}\Big) \end{aligned} \tag{13}$$

$$w_i \leftarrow w_i - \eta\left(\left(\frac{1 - S_{ab}}{2} - \frac{1}{1 + e^{(s_a - s_b)}}\right)(x_i^a - x_i^b) + \gamma_{w_i} w_i\right) \tag{14}$$

We show the general learning process of RankingFM in Algorithm 1, which can handle multi-class ranking tasks, e.g., Trec Contextual Suggestion[3] and conventional LtR scenarios. Nevertheless, as explained in Section 1, in most real-world scenarios of CF, negative examples and unknown positive examples are mixed together and hardly to be distinguished [14], known as one-class collaborative filtering (OCCF). It can be seen the algorithm has $O(|\mathcal{U}||\mathcal{A}||\mathcal{B}|)$ training triples, where $|\mathcal{A}|$ and $|\mathcal{B}|$ represent the size of observed and unobserved actions by the user $u \in \mathcal{U}$, so we have $\mathcal{A} \cup \mathcal{B} = \mathcal{I}, \mathcal{A} \cap \mathcal{B} = \varnothing$, where $|\mathcal{I}|$ represents the number of items. That is, we need to compute all the full gradient in each update step, which is infeasible because $|\mathcal{B}|$ is usually huge in practice. To solve this problem, it is natural to propose a sampling scheme (e.g., bootstrapping [20]), which, on one hand, can make the best use of unobserved feedback for the learning; on the other hand, helps to reduce the runtime of the algorithm. The slightly revised RankingFM for OCCF is shown in Algorithm 2, i.e., Line 5 in Algorithm 1 is replaced by Line 1-3 of Algorithm 2. In this case, $\boldsymbol{x}^a$ denotes the observed positive sample vector, while $\boldsymbol{x}^b$ denotes the unobserved sample vector.

In terms of the computational complexity, it can be seen that the complexity of Eq. (13) and Eq. (14) is $O(kn)$ and $O(n)$ respectively. Thus RankingFM also has a linear computational complexity for each training pair. Moreover, for a CF scenario, most elements x_i in a vector $\boldsymbol{x}$ are zero. For example, let $N(\boldsymbol{x})$ be the number of non-zero elements in the feature vector $\boldsymbol{x}$ and $\overline{N(\boldsymbol{x})}$ be the average number of non-zero elements in all vectors. We can see that $\overline{N(\boldsymbol{x})} \ll n$ under huge sparsity, i.e., the complexity becomes $O(k\overline{N(\boldsymbol{x})})$ in the CF settings.

4 Efficient Lambda Samplers

4.1 Sampling Analysis

RankingFM is made to work quite well due to the design of the pairwise CE loss function, which is fine if that is the desired loss. However, typical pairwise loss functions are devised to maximize the AUC metric, which is clearly position-independent. For item recommendations, the recommendation quality is highly position-biased because high accuracy near the top of a ranked list is more important to users. To solve this challenge, lambda-based approaches (e.g., LambdaRank [7, 15]) have been presented by incorporating ranking bias into pairwise comparison. Inspired by this idea, we may design a similar weighting term $\xi_{a,b}$ for further optimization of RankingFM, which is hereafter called Lambda Factorization Machines (LambdaFM for short). $\xi_{a,b}$[4] is designed to incorporate the size of change of a specific ranking measure by swapping two items (i.e., a and b) of this pair with different relevance levels, the way of which is called lambda (or λ). The new learning weight is defined as:

$$\lambda_{ab} = \Big(\frac{1 - S_{ab}}{2} - \frac{1}{1 + e^{(s_a - s_b)}}\Big)\xi_{ab} \tag{15}$$

[3] `https://sites.google.com/site/treccontext/`

[4] The work in [26] only adopted NDCG for the analysis of lambda whereas we here consider multiple measures.

where ξ_{ab} can be the difference of any ranking measure, e.g., NDCG, Reciprocal Rank (RR), or Average Precision (AP), computed by:

$$\xi_{ab} = \begin{cases} |N(2^{l_a} - 2^{l_b})\Big(\frac{1}{\log(1+r_a)} - \frac{1}{\log(1+r_b)}\Big)| & \text{if } \xi_{a,b} \text{ is } |\triangle NDCG_{ab}| \\ |\frac{1}{R}\left[\frac{n+1}{r_b} - \frac{m}{r_a}\right] + \sum_{k=r_b+1}^{r_a-1} \frac{l_k}{k}| & \text{if } \xi_{a,b} \text{ is } |\triangle AP_{ab}| \\ |\frac{1}{r_b} - \frac{1}{r}| & \text{if } \xi_{a,b} \text{ is } |\triangle RR_{ab}| \text{ and } r_b < r \leq r_a \end{cases} \tag{16}$$

where N is the reciprocal of maximum DCG for a user; l_a and l_b are levels of relevance for item a and b, respectively; r_a and r_b are the rank positions of a and b, respectively; n and m are the number of relevant items at the top r_b and the top r_a positions, respectively; l_k is the binary relevance label of the item at rank position k, i.e., 1 for relevance and 0 otherwise, R is the number of relevant items; r is the rank of the top relevant item in the ranking list. Note that the above equation of RR_{ab} holds only when $r_b < r \leq r_a$, otherwise there is no RR gain. We find that the above implementation is reasonable for multi-class scenarios in typical LtR tasks but impractical in OCCF settings. The reason is that to calculate ξ_{ab} it requires to compute scores of all items using Eq. (3) to obtain the rank, i.e., r_a and r_b in Eq. (16). For typical IR tasks, the candidate documents for a query in training datasets have usually been limited to a small size (e.g., 1000) because of query filtering [27]. However, for recommendation with implicit feedback, the size of candidate items is usually very huge (e.g., 10 million) as all unobserved items should be considered as candidates. Thus, the computational complexity before the update of each training pair has becomes $O(kn|\mathcal{I}|)$. In other words, the original lambda implementation for LambdaRank is not suitable for OCCF settings [26].

To bypass this complexity issue, we devise two efficient lambda-based sampling schemes in the followings. Assume we have an ideal lambda function λ_{ab}, if we have a sampling scheme that generates the training item pairs with the probability proportional to $\lambda_{ab}/(\frac{1-S_{ab}}{2} - \frac{1}{1+e^{(s_a - s_b)}})$ (just like ξ_{ab}), then we can have almost equivalent training models. Further, we give an example of a ranked list (with implicit feedback) to show which item pairs should be assigned with higher sampling weights, where +1 and -1 are positive and unobserved items, respectively.

$$\text{Rank Order}: \overbrace{-1,\ -1,\ +1,\ -1,-1,\underbrace{-1,\ -1,\ +1}_{\xi_{86}}}^{\xi_{81}}, -1, -1$$

According to Eq. (16), we calculate that ξ_{81} is 0.42, 0.54 and 0.67 when ξ_{81} is $\triangle NDCG$, $\triangle AP$ and $\triangle RR$ respectively, and that ξ_{86} is 0.02, 0.04 and 0 when ξ_{81} is $\triangle NDCG$, $\triangle AP$ and $\triangle RR$ respectively. Obviously, ξ_{81} is always larger than ξ_{86} regardless of the ranking of the positive item and which ranking measure we employ. This implies ξ_{81} is likely to be a more informative[5] training pair (compared with ξ_{86}) if the unobserved item b has a higher ranking position.

[5] In the followings, we refer to a training pair (a,b) as an informative pair if ξ_{ab} is larger after swapping a and b, the unobserved item b is called a good or informative item.

Algorithm 3 Lambda Learning Scheme I (LFM-I)

1: Uniformly draw u from $\mathcal{U}$
2: Uniformly draw a from $\mathcal{A}$
3: **repeat**
4: Uniformly draw b from $\mathcal{I}\backslash\mathcal{A}$
5: Generate a random variable $\rho_{\mathbf{rand}}(u,a,b) \in [0,1]$
6: Calculate the utility function $\rho(u,a,b) = \frac{e^{-s_{ab}}}{1+e^{-s_{ab}}}$
7: **until** $\rho_{\mathbf{rand}} \leq \rho(u,a,b)$

Algorithm 4 Lambda Learning Scheme II (LFM-II)

1: **Require:** Unobserved item set $\mathcal{I}\backslash\mathcal{A}$, parameter ρ and m, scoring function $\hat{y}(\cdot)$
2: Sample a rank r from the power law distribution $pr(r(b)) \propto (\frac{1}{r(b)+1})^{2\rho}$, $\rho \in [0,1]$, where $r(b)$ (starting from 0) is the rank of item b, and ρ is a coefficient that can be tuned for the optimal results. Note that LFM-II will reduce to RankingFM when $\rho = 0$.
3: Uniformly draw b_1,...,b_m from $\mathcal{I}\backslash\mathcal{A}$
4: Compute $\hat{y}(\boldsymbol{x}^{b_1})$,...,$\hat{y}(\boldsymbol{x}^{b_m})$, and then sort b_1,...,b_m by descending order of $\hat{y}(\boldsymbol{x}^{b_1})$,...,$\hat{y}(\boldsymbol{x}^{b_m})$
5: Return one item b, which is currently ranked on the r-th position.

Based on this insightful finding, we believe the item pairs whose unobserved item has a higher rank should be drawn with higher probability. This is because the top ranked unobserved items hurt the ranking performance more than those with lower ranked positions [27, 28]. With the intuitive observation and above analysis, we devise two simple yet effective sampling schemes to further optimize RankingFM for top-N item ranking.

4.2 Lambda-based Learning Schemes

Scheme I According to the above analysis, we argue that the item pair $(8,1)$ is supposed to be sampled with higher probability than the $(8,6)$ pair. In addition, we observe that the value of s_{81} is smaller than that of s_{86} because

$$\begin{aligned} s_{81} &= \hat{y}(\boldsymbol{x}^8) - \hat{y}(\boldsymbol{x}^1) \\ s_{86} &= \hat{y}(\boldsymbol{x}^8) - \hat{y}(\boldsymbol{x}^6) \\ \hat{y}(\boldsymbol{x}^1) &> \hat{y}(\boldsymbol{x}^6) \end{aligned} \tag{17}$$

The above observation suggests that we should sample more item pairs with small preference difference, such as s_{81}. We thus propose an intuitive learning scheme with a dynamic utility function $\rho(u,a,b)$ to judge whether a (u,a,b) triple contains an informative training pair (or a good negative item) such that swapping the positions of a and b could lead to a larger change of a desired ranking loss. The lambda-motivated learning scheme is shown in Algorithm 3, where ρ is a sigmoid function. Hereafter we denote the new algorithm (replacing Line 5 of Algorithm 1 with Algorithm 3) as LFM-I. It can be clearly seen that a smaller s_{ab} will contribute to a larger utility $\rho(u,a,b)$. In terms of the computational complexity, the complexity to calculate the original ξ_{ab} is $O(k\overline{N(\boldsymbol{x})}|\mathcal{I}|)$, while the complexity of Algorithm 3 is $O(k\overline{N(\boldsymbol{x})}T)$, where T is the size of sampling trials. In general, we have $T \ll |\mathcal{I}|$ in the beginning of the training and $T < |\mathcal{I}|$ when the training reaches convergence. The reason is because in the beginning, the elements of matrix $\boldsymbol{V}$ are initialized by a standard normal distribution with mean 0 and variance 0.1 (see Algorithm 1), and thus the distribution of s_{ab} also

follows an approximate normal distribution with mean 0. In this case, it is quick to find an unobserved item that meets the condition (i.e., $\rho_{\mathbf{rand}} \leq \rho(u, a, b)$) as most possible values of $\rho(u, a, b)$ are around 0.5. After several training round, most observed items are likely to ranked higher than the unobserved items (i.e., $s_{ab} > 0$), and thus $\rho(u, a, b)$ is likely to be smaller than 0.5, which will lead to a bit larger T. However, it is impossible that all positive items are ranked higher than unobserved items, so in general we still have $T < |\mathcal{I}|$ with $\rho_{\mathbf{rand}}(u, a, b) \in [0, 1]$.

Scheme II According to Section 4.1, a straightforward sampling scheme with the same training effect of the original lambda can be implemented by calculating scores of all items to obtain the possible ranks, and then oversample higher ranked unobserved items. Unfortunately, this learning scheme has the same computational complexity with the original lambda strategy, which is clearly infeasible in practice. To overcome this issue, we first employ a uniform sampling to select m candidates. Then we compute the scores of these candidate items to achieve possible rank orders, and sample the rank by a power-law distribution $pr(r)$ (In practice, $pr(r)$ can be replaced with other distributions, such as exponential and linear distributions as long as $pr(r)$ meets the condition that assigns larger sampling weight to top ranked unobserved items.). Because the first sampling is uniform, the sampling probability density for each item has almost equivalent effect with that from the original (expensive) global sampling. The proposed sampler is shown in Algorithm 4. We refer to RankingFM with the seconding learning scheme as LFM-II. The complexity before performing each pairwise comparison reduces to $O(mkn + m \log m)$, where m is often set to a small value (e.g., $m = 20, 50$). Therefore, by implementing scheme II, we are also able to find an efficient way to bypasses the expensive computational complexity.

5 Experiments

We conduct a set of experiments to evaluate the top-N recommendation accuracy of RankingFM and LambdaFM, compared to several state-of-the-art methods.

5.1 Settings

We use three real-world CF datasets to verify the performance of our proposed methods, namely Libimseti.cz[6] (user-user pairs, where the users recommended as daters are regarded as items here), Lastfm[7] (user-music-artist tuples) and Yahoo[8] (user-music-artist-album tuples). In order to speed up the experiments, we follow the common practice as in [5] by randomly sampling a subset of users from the Libimseti and Yahoo datasets, and a subset of items from the Lastfm dataset. Table 1 summarizes the statistics of the three datasets used in this work. We evaluate the results of top-N item recommendations by two standard metrics, namely, Precision@N and Recall@N (denoted by Pre@N and Rec@N,

[6] http://www.occamslab.com/petricek/data/

[7] dtic.upf.edu/~ocelma/MusicRecommendationDataset/lastfm-1K.html

[8] http://webscope.sandbox.yahoo.com/catalog.php?datatype=r&did=2

Table 1: Basic Statistics of Datasets.

DataSets	#Users	#Items	#Records	Density	Rsize	Csize	#Artists	#Albums
Libimseti	5000	82444	642454	0.16%	128.49	7.79	-	-
Lastfm	983	60000	246853	0.42%	251.12	4.11	25147	-
Yahoo	2450	124346	911466	0.29%	372.03	7.33	9040	19851

The "Rsize" and "Csize" columns are the average number of records (e.g., ratings) for each user and for each item respectively.

respectively) [12], where N is the number of recommended items. Details about the two metrics are omitted for saving space.

In our experiments, we compare our methods with four powerful baseline methods: Most Popular(MP) [20], Factorization Machines(FM) [17], Bayesian Personalized Ranking with matrix factorization (BPR) [20], Pairwise Interaction Tensor Factorization (PITF)[9] [22]. Note we adapt FM for the top-N recommendation task by binarizing rating values[10] (denoted as FMB). Since the frequency of a user listening to a song (i.e., relevance feedback) can be obtained from the Lastfm dataset, we also recommend songs by leveraging such information (denoted as FMF[11]). Note that the frequency information has a large range compared with ratings (e.g., $[1, 5]$ interval). For example, a user may listen to a song in hundreds of times. In this paper, we employ a trivial function $\frac{1}{1+f^{-1}}$ to map the frequency into $[0.5, 1)$, where f represents the frequency. Besides, for a fair comparison, we also exploit the same bootstrap sampling as in BPR to make use of the large number of unobserved items.

All factorization models use a factorization dimension of k=30. Results for k=10, 50, 100 give consistent conclusion but are omitted due to space limitations. In terms of η and γ_θ, we apply the 5-fold cross-validation to find optimal values for BPR. For PITF, our results show that it performs best with the same η and γ_θ of BPR. For FM (FMB and FMF), we apply the same method to tune η and γ_θ individually; For RankingFM and LambdaFM, we use the same η and γ_θ with BPR for comparison. Specifically, η is set to 0.01 on the Libimseti and Yahoo datasets, and 0.08 on the Lastfm dataset; γ_θ (including γ_{w_i}, $\gamma_{v_{i,f}}$) is set to 0.01 on Libimseti dataset, and 0.05 on Lastfm and Yahoo datasets. Note that we find that all FM based models perform well enough by just using polynomial term (see Eq. (1)). $\rho \in [0, 1]$ is specific for LFM-II, which is discussed later.

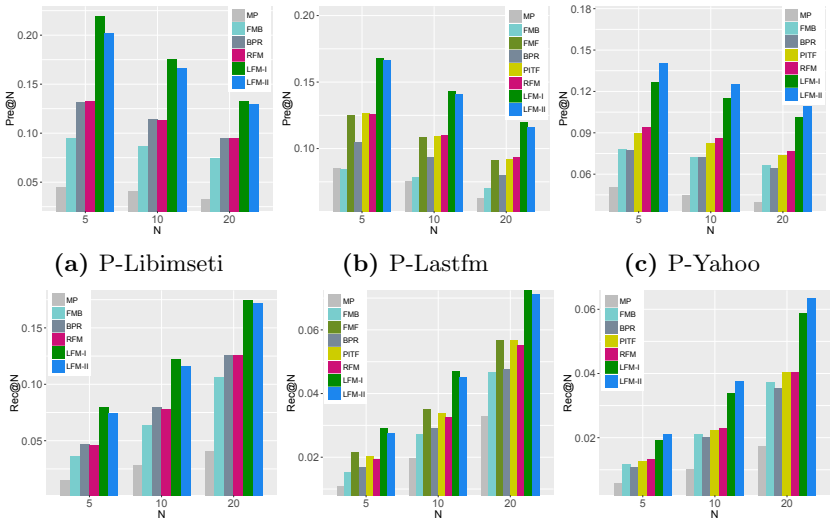
5.2 Results

Accuracy Analysis Figure 1(a-f) shows the top-N recommendation accuracy of all algorithms on the three datasets. First, we clearly observe that our proposed RankingFM (RFM) largely outperforms the original FM model (i.e., FMB), which empirically indicates the pairwise approach outperforms the pointwise approach with the common 0/1 interpretation [20]. The reason is because the two algorithms have the same scoring function but only differ in loss functions:

[9] Due to lack of contexts, PITF is not applicable to the Libimseti dataset.

[10] It is a standard way to solve the one-class problem in CF [14].

[11] FMF is identical to FMB in the other datasets, since the frequency of all observed actions is 1.



(a) P-Libimseti **(b)** P-Lastfm **(c)** P-Yahoo

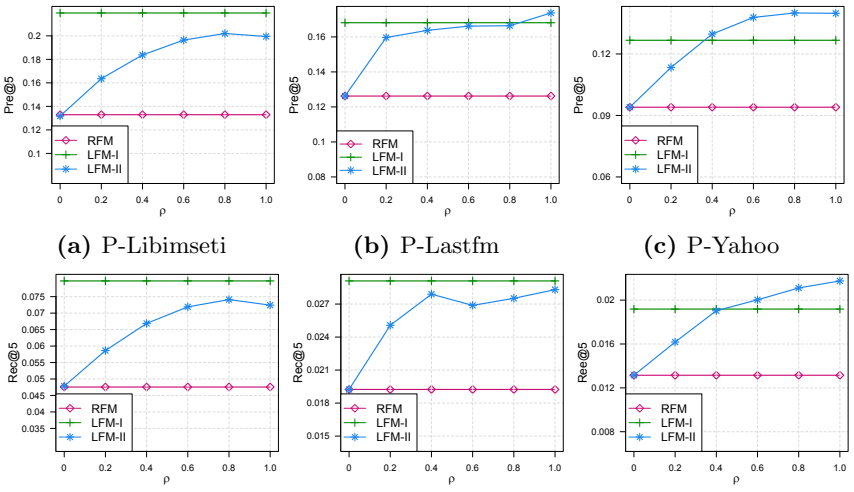
(d) R-Libimseti **(e)** R-Lastfm **(f)** R-Yahoo

Fig. 1: Performance comparison w.r.t. top-N values, i.e., Pre@N (P) and Rec@N (R). ρ is fixed to 0.8 for LFM-II, and m is fixed to 50.

FMB applies the pointwise square loss, while RFM uses the pairwise CE loss for optimization. Second, the proposed LambdaFM (LFM-I, LFM-II) consistently outperforms other methods in terms of both ranking metrics. This is because LambdaFM (1) directly optimizes the ranking metrics by the design of two lambda-based sampling schemes (vs. BPR, PITF, FMB, FMF and RFM); (2) estimates more accurate ordering relations between candidate items by incorporating additional contextual variables (e.g., artists and albums) (vs. BPR and PITF). Third, we find that RFM achieves almost the same results with BPR and PITF on the Libimseti and Lastfm datasets. The reason is because all the three approaches exploit the pairwise loss function but with different prediction functions. FM (from RFM) is identical to matrix factorization (from BPR) with user-item feature vector and tensor factorization (from PITF) with user-item-artist feature vector. In other words, RFM is able to mimic state-of-the-art ranking algorithms (i.e., BPR and PITF) by feature engineering. Fourth, FMF performs much better than FMB on the Lastfm dataset. This indicates that a user's preference to a song can be inferred more accurately by leveraging playing times information. The intuition is that the more times she played a music track, the higher preference she expresses implicitly. Several other insights can be obtained in Figure 1 but are omitted for space reasons.

Tuning ρ Figure 2 illustrates the impact of ρ for learning scheme II in terms of Pre@5 and Rec@5[12]. First, by assigning a larger ρ, it is easy to find LFM-II noticeably outperforms RFM. The better results indicate that the lambda-motivated sampler (scheme II) works effectively to deal with the suboptimal

[12] The results w.r.t. other top-N values are consistent, but are omitted for saving space.



(a) P-Libimseti **(b)** P-Lastfm **(c)** P-Yahoo

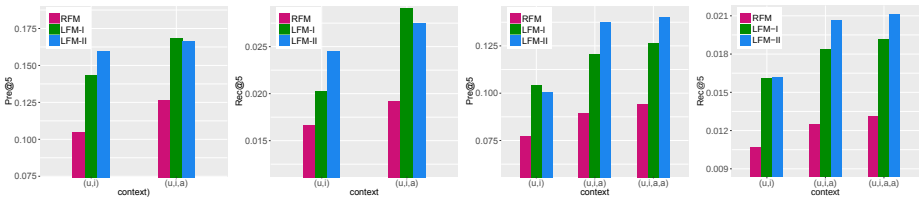
(d) R-Libimseti **(e)** R-Lastfm **(f)** R-Yahoo

Fig. 2: Parameter tuning for LFM-II w.r.t. Pre@5 (P) and Rec@5 (R). $\rho \in \{0, 0.2, 0.4, 0.6, 0.8, 1.0\}$, $m = 50$.

results of pairwise ranking. Note LFM-II is equivalent to RFM when $\rho = 0$ according to Algorithm 4. Second, the performance of LFM-II on all three datasets increases with the growth of ρ. In particular, on the Libimseti dataset, the performance achieves the optimal value when $\rho = 0.8$, and starts to decrease when $\rho = 1.0$. The reason is because only several top ranked items have the chance to be selected as candidates for the pairwise comparison when $\rho = 1.0$, and in this case, many unobserved items will not be seen by the learning algorithm, which probably leads to relatively worse performance because of insufficient training samples. On the other side, its performance has not obviously decreased when ρ is set to 1.0 on the Lastfm and Yahoo datasets, which suggests that (1) LFM-II may work well even by picking the top from m randomly selected items; (2) the performance is expected to be improved further by setting a larger sampling size (i.e., m)[13].

Impact of Context We compare the performance changes of RankingFM and LambdaFM by gradually adding additional contextual variables. First, Figure 3(a-d) indicates that both RankingFM and LambdaFM with (u, i, a) noticeably outperform that with (u, i) tuples on both datasets. Second, we can see RankingFM and LambdaFM with (u, i, a, a) tuples outperform that with (u, i, a) tuples from Figure 3(c-d). The intuition behind is that a use's preference to a music track can be inferred more accurately by taking into account of the artist and album information. Hence, we argue that in general by adding useful context features, our models are able to obtain significant recommendation improvements.

[13] Note that a larger sampling size m will result in a larger computational complexity.



(a) P-Lastfm **(b)** R-Lastfm **(c)** P-Yahoo **(d)** R-Yahoo

Fig. 3: Performance comparison w.r.t. Pre@5 (P) & Rec@5 (R) by adding context. ρ is fixed to 0.8, and m is fixed to 50 for LFM-II. (u, i) is a user-item (i.e., music) pair and (u, i, a) is a user-item-artist tuple in (a-d); (u, i, a, a) is a user-item-artist-album tuple in (c) and (d).

6 Conclusion and Future Work

In this paper, we have introduced two ranking predictors, namely RankingFM and LambdaFM. In contrast to other CF algorithms, RankingFM and LambdaFM are general context-aware recommendation algorithms that are able to incorporate any types of context information. Besides, we design two intuitive sampling schemes for LambdaFM, with which LambdaFM is made more reasonable for optimizing item ranking in OCCF settings. Our experiments on three public CF datasets show that RankingFM and LambdaFM performs better than several state-of-the-art CF methods. In particular, LambdaFM (with two proposed sampling schemes) demonstrates superior ranking performance in the top-N item recommendation task, reflected in two standard ranking metrics.

For future work[14], we plan to (i) develop more advanced samplers to improve LambdaFM without negative effects on efficiency; (ii) investigate the generalization of the suggested lambda strategies on other well-known pairwise loss functions, e.g., hinge Loss [8], exponential loss [4] as well as fidelity loss [13] (iii) investigate the performance of both RankingFM and LambdaFM for traditional multi-class ranking tasks, such as web search and Trec Contextual Suggestion.

References

1. C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *ICML*, pages 89–96, 2005.
2. C. J. Burges. From ranknet to lambdarank to lambdamart: An overview.
3. Z. Cao, T. Qin, T. Liu, M. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *ICML*, pages 129–136, 2007.
4. W. Chen, T.-Y. Liu, Y. Lan, Z.-M. Ma, and H. Li. Ranking measures and loss functions in learning to rank. In *NIPS*, pages 315–323, 2009.
5. K. Christakopoulou and A. Banerjee. Collaborative ranking with a push at the top. In *WWW*, pages 205–215, 2015.
6. P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *RecSys*, pages 39–46, 2010.

[14] We would refer the interested reader to [26] for a detailed analysis about LambdaFM.

7. P. Donmez, K. M. Svore, and C. J. Burges. On the local optimality of lambdarank. In *SIGIR*, pages 460–467, 2009.
8. R. Herbrich, T. Graepel, and K. Obermayer. Support vector learning for ordinal regression. 1999.
9. Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM*, pages 263–272, 2008.
10. A. Karatzoglou, X. Amatriain, L. Baltrunas, and N. Oliver. Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *RecSys*, pages 79–86, 2010.
11. Y. Koren. Collaborative filtering with temporal dynamics. pages 89–97, 2010.
12. X. Li, G. Cong, X.-L. Li, T.-A. N. Pham, and S. Krishnaswamy. Rank-geofm: a ranking based geographical factorization method for point of interest recommendation. In *SIGIR*, pages 433–442, 2015.
13. M.Tsai, T.Liu, T.Qin, H.Chen, and W.Ma. Frank: a ranking method with fidelity loss. In *SIGIR*, pages 383–390, 2007.
14. R. Pan, Y. Zhou, B. Cao, N. N. Liu, R. Lukose, M. Scholz, and Q. Yang. One-class collaborative filtering. In *ICDM*, pages 502–511, 2008.
15. C. Quoc and V. Le. Learning to rank with nonsmooth cost functions. 19:193–200, 2007.
16. S. Rendle. Factorization machines. In *ICDM*, pages 995–1000, 2010.
17. S. Rendle. Factorization machines with libFM. *TIST*, pages 57:1–57:22, 2012.
18. S. Rendle, L. Balby Marinho, A. Nanopoulos, and L. Schmidt-Thieme. Learning optimal ranking with tensor factorization for tag recommendation. In *SIGKDD*, pages 727–736, 2009.
19. S. Rendle and C. Freudenthaler. Improving pairwise learning for item recommendation from implicit feedback. In *WSDM*, pages 273–282, 2014.
20. S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. BPR: bayesian personalized ranking from implicit feedback. In *UAI*, pages 452–461, 2009.
21. S. Rendle, Z. Gantner, C. Freudenthaler, and L. Schmidt-Thieme. Fast context-aware recommendations with factorization machines. In *SIGIR*, pages 635–644, 2011.
22. S. Rendle and L. Schmidt-Thieme. Pairwise interaction tensor factorization for personalized tag recommendation. In *WSDM*, pages 81–90, 2010.
23. R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *NIPS*, volume 20, pages 1257–1264, 2008.
24. Y. Shi, A. Karatzoglou, L. Baltrunas, M. Larson, A. Hanjalic, and N. Oliver. TFMAP: optimizing map for top-n context-aware recommendation. In *SIGIR*, pages 155–164, 2012.
25. Y. Shi, A. Karatzoglou, L. Baltrunas, M. Larson, N. Oliver, and A. Hanjalic. CLiMF: learning to maximize reciprocal rank with collaborative less-is-more filtering. In *RecSys*, pages 139–146, 2012.
26. F. YUAN, G. Guo, J. Jose, L. Chen, H. Yu, and W. Zhang. Lambdafm: Learning optimal ranking with factorization machines using lambda surrogates. In *CIKM*, 2016.
27. W. Zhang, T. Chen, J. Wang, and Y. Yu. Optimizing top-n collaborative filtering via dynamic negative item sampling. In *SIGIR*, pages 785–788, 2013.
28. H. Zhong, W. Pan, C. Xu, Z. Yin, and Z. Ming. Adaptive pairwise preference learning for collaborative recommendation with implicit feedbacks. In *CIKM*, pages 1999–2002, 2014.