BoostFM: Boosted Factorization Machines for Top-N Feature-based Recommendation

Fajie Yuan[†], Guibing Guo[‡], Joemon M. Jose[†], Long Chen[†], Haitao Yu[⊤†], Weinan Zhang[⊥]
[†]University of Glasgow, UK [‡]Northeastern University, China
[⊤]University of Tsukuba, Japan [⊥]Shanghai Jiao Tong University, China
f.yuan.1@research.gla.ac.uk, guogb@swc.neu.edu.cn, wnzhang@sjtu.edu.cn

ABSTRACT
Feature-based matrix factorization techniques such as Factorization Machines (FM) have been proven to achieve impressive accuracy for the rating prediction task. However, most common recommendation scenarios are formulated as a top-N item ranking problem with implicit feedback (e.g., clicks, purchases) rather than explicit ratings. To address this problem, with both implicit feedback and feature information, we propose a feature-based collaborative boosting recommender called *BoostFM*, which integrates boosting into factorization models during the process of item ranking. Specifically, BoostFM is an adaptive boosting framework that linearly combines multiple homogeneous component recommenders, which are repeatedly constructed on the basis of the individual FM model by a re-weighting scheme. Two ways are proposed to efficiently train the component recommenders from the perspectives of both pairwise and listwise Learning-to-Rank (L2R). The properties of our proposed method are empirically studied on three real-world datasets. The experimental results show that BoostFM outperforms a number of state-of-the-art approaches for top-N recommendation.

ACM Classification Keywords
1.2.6 Artificial Intelligence: Learning (K.3.2)

Author Keywords
Factorization Machines; BoostFM; Pairwise; Listwise; Feature-based; Top-N Ranking

INTRODUCTION
Recently, matrix factorization (MF) models have gained much attention in collaborative filtering (CF) recommender systems due to their good performance and efficiency when dealing with large sparse datasets [4]. However, classical MF methods only involve the interactions between users and items but no consideration of additional auxiliary information, such as the context and item side information. In practice, it is common that auxiliary information is available in recommender systems. Thus, we may regard such information as features to help improve recommendation quality, which is hereafter referred to as Feature-based Recommendation[1]. As a matter of fact, many MF variants have been proposed in recent literature [13, 14]. Most of these models rely on strict assumptions and usually require complicated inference when being adapted for a new problem, though they are designed to incorporate certain types of feature information in specific settings. Therefore, the models capable of integrating any types of auxiliary information are more practical, as well as more elegant in theory. So far, two of the most flexible and effective methods for feature representation in CF are based on Tensor Factorization (TF) [12, 27] and Factorization Machines (FM) [21, 4]. Generally, the type of feature information used in TF is usually limited to categorical variables. By contrast, FM takes as input both categorical and real-valued variables, and thus is more general.

It has been recognized that the original FM model is presented for the rating (or other ordering information on user feedback) prediction task, the performance of which is usually evaluated by the metric Root Mean Square Error (RMSE). However, achieving good accuracy in terms of RMSE usually does not guarantee equivalent effectiveness in the case of top-N item recommendation [28]. On the other hand, in practice, most prevalent user feedback is not explicit (e.g., in terms of ratings) but implicit [23]. Examples for implicit feedback are clicks, purchases, watched videos or played songs, which are much cheaper to obtain since a user does not have to express his taste explicitly. Implicit feedback is often one-class [22], i.e., only positive class is available, and thus algorithms optimized for multiple classes cannot be directly applied for implicit feedback data [31, 32]. In this paper, we study the problem of optimizing item ranking with implicit feedback and feature information.

To deal with both feature information and implicit feedback, we employ the boosting technique to improve the recommendation accuracy by combing the power of multiple individual 'weak recommender'. Boosting techniques were first employed to improve the performance of classification by integrat-

IUI 2017, March 13-16, 2017, Limassol, Cyprus

DOI: http://dx.doi.org/10.1145/3025171.3025211

[1] In this regard, the classical user-item based recommendation can be treated as the simplest form of Feature-based Recommendation (where users and items are read as two basic features). Note that we target at modeling more general features in this work.

ing a series of weak classifiers (i.e., the classification accuracy rate should be larger than 0.5) into a stronger one with better performance [11]. Previous research has proven that boosting techniques usually come with better convergence properties and stability [2, 5]. So far, the most common implementation of boosting is AdaBoost [8], although some newer boosting variants are reported. We find that boosting techniques have been recently introduced to solve recommendation problems with better results than single CF algorithms [11, 28, 16, 5]. However, we emphasize that all existing solutions are based on the basic matrix factorization model, which fails to incorporate more general feature information. Moreover, in our work the learning process of each component recommender is optimized for (item) ranking with implicit feedback, which differs from previous works either optimized for rating prediction [11] or item ranking based on explicit rating datasets [5, 28]. Finally, we formulate feature information as context and item (with side information) vectors, the flexible representation of which allows to describe various input formats such as user profile, item attributes, successive context as well as their combinations. Thanks to this, in our boosting procedure each observed context-item pair is able to be treated as a training instance for weighting calculation, which makes our work distinguished from other literature either treating a given user or query as an instance.

In this paper, we propose BoostFM, a boosting approach for top-N Feature-based Recommendation with implicit feedback, by combining the well-known boosting framework with FM. Specifically, we first employ FM to build the individual component recommender and multiple homogeneous component recommenders are linearly combined to form a strong recommender. The coefficient of each component recommender is calculated from a weighting function designed based on a certain performance metric (e.g., AUC and NDCG). At each boosting round, we devise two optimization methods to train our component recommenders inspired by ideas of both pairwise and listwise Learning-to-Rank (L2R). In addition, we develop a re-weighting strategy and assign a dynamic weight to force the optimization concentrating more on observed context-item interactions with bad evaluation performance. Finally, we perform experiments on three publicly available datasets and compare BoostFM with state-of-the-art CF approaches. Our results demonstrate that BoostFM noticeably outperforms all counterparts in terms of top-N recommendation accuracy.

RELATED WORK

The work in this paper closely relates to two research areas, i.e., top-N Feature-based Recommendation and Boosting techniques. We discuss them separately and then position our work with respect to them.

Feature-based Recommendation

Feature-based Recommendation has gained much popularity in recent years, and considerable efforts have been made by researchers and practitioners. Prior works (e.g., [1]) exploited feature information for pre- or post-filtering to make standard models feature-aware, where potential (e.g., 2-order or even high order) interactions between different feature variables have not been considered, leading to unsatisfactory performance. Recent works mostly revolve around combining feature information with factorization to directly model the user, item and auxiliary feature variables [26, 31]. Two significant approaches have been developed: tensor factorization (TF) [12, 29] and Factorization Machines (FM) [21, 4]. Whereas those works are designed to solve the rating prediction task [22, 31], the goal of item recommendation is usually regarded as a top-N item ranking task. Motivated by this, state-of-the-art methods by combining L2R and feature-based CF models have been proposed with significant accuracy improvements [26, 27, 18]. For instance, TFMAP [27] employs TF to capture the 3-way user-item-context relations, and learns it by maximizing Mean Average Precision (MAP); in the same light, CARS explores multiple objective functions to train a novel TF model. Ranking FM [18, 31, 32, 10], on the other side, aims to exploit FM as the rating function to model the pairwise feature interaction, and to build the ranking algorithm by maximizing various ranking measures such as the Area Under the ROC Curve (AUC) and the Normalized Discount Cumulative Gain (NDCG).

Boosting

Boosting technique is a general framework for improving the accuracy of a given learning algorithm [8, 30]. The central idea is to repeatedly form a number of 'weak learners' by using a homogeneous weak algorithm on re-weighting training data. Then, a strong learner with boosted total performance is created by composing weak learners linearly. Boosting was originally proposed to enhance the performance of binary classification, where AdaBoost (Adaptive Boosting) is the most well-known boosting algorithm. Following this, various extensions have been made to deal with problems of multiclass classification [9], regression [2], and ranking [30].

Recently, researchers have proposed to construct boosting techniques in Recommender Systems. For example, two boosting frameworks inspired by AdaBoost have been presented for the rating prediction task by applying both memory- and model-based CF algorithms [11]. AdaMF [28] borrows the idea from adaRank by combining matrix factorization (MF) recommender with boosting methods. The coefficient function for each MF recommender is calculated based on the Normalized Discount Cumulative Gain (NDCG) performance of the stronger recommender. However, the component recommenders are constructed using the CF algorithm for rating prediction, which is suboptimal for item recommendation task. Similar work has been done in [5], where the component recommender is constructed using Probability matrix factorization (PMF) on explicit rating datasets.

Our work relates to above works, but differs in several significant aspects. First, in BoostFM, the component recommender is constructed by feature-based factorization models instead of a simple scoring function (i.e., so-called weak learner). Again individual FM model can easily achieve relatively good recommendation. Thus, we can regard FM as a relatively strong recommender[2]. Second, the component recommenders are

[2]Previous literature has shown that AdaBoost demonstrates better generalizing performance with correlated strong learners [15].

constructed by optimizing weighted pairwise and 'listwise' objective functions in the form of implicit feedback, rather than approximating users' explicit ratings (e.g., [11, 28]). Third, most above boosting techniques either treat a query [30], a given user [28], or a feature vector [8] (but with explicit ordering information) as a unit to assign boosting instance weights, the way of which might not be directly applied for pairwise item comparisons with both implicit feedback and auxiliary information. To address this, we formulate a flexible feature representation for describing various auxiliary information in FM, whereby an observed context-item pair can be treated as a training instance for boosting weight calculation. With these advantages, our BoostFM framework and component recommenders by two ways of optimization can be coupled together seamlessly. To our best knowledge, BoostFM is the first study for feature-based collaborative ranking by adopting the boosting technique. Note it is worth mentioning that one recent work [4] has exploited the gradient boosting algorithm for rating prediction with FM. However, we argue that our work targets at top-N item ranking, a completely different recommendation task.

PRELIMINARIES

In this section, we briefly describe a key component related to our BoostFM, i.e., Factorization Machines (FM), and a flexible feature vector representation for different auxiliary information. Then we present the problem formulation of Feature-based recommendation from implicit feedback.

Factorization Machines

Factorization Machines (FM) [21] is a cutting-edge feature-based CF algorithm, which utilizes a factorized representation to model the nested interactions among n input variables in feature vector $\mathbf{x}$[3]. In this subsection, we will show how to employ FM to represent general feature information.

Let $\mathbf{x}_c \in \mathbb{R}^{VC}$ be an arbitrary feature vector that represents context $c \in C$ with VC real-valued variables, and $\mathbf{x}_i \in \mathbb{R}^{VI}$ be a feature vector that represents item $i \in I$ with VI variables (including both item and side information), and thus we have $\mathbf{x} = (\mathbf{x}_c, \mathbf{x}_i)$. For example, in a music recommender system $\mathbf{x}$ could be represented as

$$\mathbf{x} = (\overbrace{\underbrace{0,...,1,...,0}_{\text{users}},\underbrace{0,0.1,...,0.1,0}_{\text{10 social friends}},\underbrace{0,1,0}_{\text{time}}}^{c},\overbrace{\underbrace{0,...,1,...,0}_{\text{music tracks}},\underbrace{0,1,...,0}_{\text{artists}}}^{i})$$

The commonly used 2-order FM is defined as

$$\hat{y}(c,i) = \hat{y}(\mathbf{x}) = \underbrace{\sum_{k=1}^{n} w_k x_k}_{\text{linear}} + \underbrace{\sum_{k=1}^{n}\sum_{k'=k+1}^{n} \langle \mathbf{v}_k, \mathbf{v}'_k \rangle x_k x'_k}_{\text{polynomial}} \tag{1}$$

where the model parameters Θ that have to be estimated are $\mathbf{w} \in \mathbb{R}^n$ ($n = VC + VI$), $\mathbf{V} \in \mathbb{R}^{n \times k}$, and $\langle \cdot, \cdot \rangle$ is the dot product of two latent vectors with the dimensionality d in the low rank space

$$\langle \mathbf{v}_k, \mathbf{v}_{k'} \rangle = \sum_{f=1}^{d} v_{k,f} \cdot v_{k',f} \tag{2}$$

where $\mathbf{v}_k$ is the k-th row vector of $\mathbf{V}$ with d factors. The linear term in Eq. (1) contains unary interactions of each feature variable x_k with the target $\hat{y}$; the polynomial term models the interaction between the k-th and k'-th latent vector with a factorized parametrization. In [20], it has shown that the efficiency of FM can be reduced to linear complexity $O(dn)$ as Eq. (1) can be mathmatically expressed as

$$\hat{y}(\mathbf{x}) = \sum_{k=1}^{n} w_k x_k + \frac{1}{2}\sum_{f=1}^{d}((\sum_{k=1}^{n} v_{k,f} x_k)^2 - \sum_{i=1}^{n} v_{k,f}^2 x_k^2) \tag{3}$$

In CF scenarios, most elements x_k in $\mathbf{x}$ are 0. Let $\overline{N(\mathbf{x})}$ be the average number of non-zero elements in all vectors, we notice $\overline{N(x)} \ll n$ under sparsity (i.e., the complexity of FM becomes $O(d\overline{N(\mathbf{x})})$). Hence, we argue that exploiting FM to train a set of component recommenders enjoys the advantage of low computational complexity, and is thus feasible in practice.

Feature-based Recommendation from Implicit Feedback

Let C be a set of context and I a set of items. In our scenario a set of observed interactions (i.e., so-called implicit feedback) $S \subset C \times I$ are available. For example, C could be a set of users, and I a set of music tracks, and S represents which music tracks a user has played, i.e., (a set of) user-music interactions. As previously mentioned, C can be fit with more general examples with additional variables, such as user mood, social friends, as well as spatial-temporal environments, denoted by $\mathbf{x}_c$, also I might express items with additional side information, e.g., the artist or category of a music track, denoted by $\mathbf{x}_i$. Note that non-observed interactions do not explicitly indicate an item, e.g., j, is not relevant to c [22]. It may be the result that the existence of j is unknown to c. The task of top-N recommendation with implicit feedback is to recommend a list of items that are supposed to be most relevant for a given context but has not interacted with before. Accordingly, for Feature-based Recommendation, the task can be formulated as: estimating a ranking $\hat{r}(j|c)$ for each non-observed (c, j) feature vector $\mathbf{x}$ (i.e., $(\mathbf{x}_c, \mathbf{x}_j)$), where $\hat{r}(j|c)$ is usually modeled by a scoring function $\hat{y}(j|c)$ (or $\hat{y}(c, j)$ interchangeably).

For item recommendation tasks, the accuracy of a recommender can be assessed by various ranking metrics, such as AUC[4] [23], NDCG [17], the Mean Average Precision (MAP) [27], Precision@N and Recall@N [14], where N is the number of recommended items. For better understanding the following approach, we show the definitions of AUC and NDCG (per context) as below

$$\text{AUC(c)} = \frac{1}{|I_c^+|}\sum_{i \in I_c^+}\frac{1}{|I \backslash I_c^+|}\sum_{j \in I \backslash I_c^+} \mathbb{I}(\hat{r}(i|c) < \hat{r}(j|c)) \tag{4}$$

$$\text{NDCG(c)} = Z_c \sum_{\hat{r}(i|c)=1}^{|I|} \frac{2^{rel_{\hat{r}(i|c)}} - 1}{\log_2(\hat{r}(i|c)+1)} \tag{5}$$

where I_c^+ is the set of items that have been observed under context c ($I \backslash I_c^+$ is the remaining items), and $\mathbb{I}(\cdot)$=1 if the condition is true, and 0 otherwise; $rel_{\hat{r}}$ represents the relevance score of a candidate item at the position $\hat{r}$, here we use a binary value 0-1 (irrelevant-relevant). Z_c is calculated from the normalization constant so that the ideal ranker will get NDCG of 1.

[3] In the followings, scalar variables are set in the default math font, e.g., w_k, $v_{k,f}$, while vectors (lower case) and matrices (upper case) are in bold face, e.g., $\mathbf{w}$, $\mathbf{V}$, $\mathbf{Q}$.

[4] Note that maximizing a smoothed AUC is still a popular way for item recommendation problem (e.g., [26, 22]), although it is position-independent.

BOOSTED FACTORIZATION MACHINES

Inspired by the AdaBoost [7, 25] algorithm for classification, we propose a novel algorithm to solve the recommendation problem by optimizing the item ranking. The algorithm is referred to as Boosted Factorization Machines (BoostFM for short).

BoostFM

We aim at devising a set of 'weak learner'[5] sequentially to model the pairwise interactions between various feature variables. Besides, the BoostFM algorithm is supposed to concentrate hard on optimizing the objective function defined based on ranking measures. We observe from Eq. (4) and Eq. (5) that the accuracy of a recommender model is determined by the rank positions (i.e. $\hat{r}(i|c)$) of positive items $i \in I_c^+$ of each context c. Thus, we devise a general performance measure function $E\left[\hat{r}(c,i,g)\right]$ to denote the recommendation accuracy associated with each observed context-item pair. The argument of general function $\hat{r}(c,i,g)$ is the rank position of item i for each context c, calculated by the trained function g. Thus we can rewrite the ranking metric of AUC and NDCG as below

$$\text{AUC}=\frac{1}{|C|}\sum_{c\in C}\frac{1}{|I_c^+|}\sum_{i\in I_c^+}E[\hat{r}(c,i,g)]=\frac{1}{|C|}\sum_{(c,i)\in S}\frac{1}{|I_c^+|}E\left[\hat{r}(c,i,g)\right] \tag{6}$$

where

$$E\left[\hat{r}(c,i,g)\right]=\frac{1}{|I\backslash I_c^+|}\sum_{j\in I\backslash I_c^+}\mathbb{I}(\hat{r}(i|c)<\hat{r}(j|c)) \tag{7}$$

$$\text{NDCG}=\frac{1}{|C|}\sum_{c\in C}Z_c\sum_{\hat{r}(i|c)=1}^{|I|}\frac{2^{rel_{\hat{r}(i|c)}}-1}{\log_2\left(\hat{r}(i|c)+1\right)}=\frac{1}{|C|}\sum_{(c,i)\in S}Z_cE\left[\hat{r}(c,i,g)\right] \tag{8}$$

where

$$E\left[\hat{r}(c,i,g)\right]=\frac{1}{\log_2\left(\hat{r}(i|c)+1\right)} \tag{9}$$

To maximize Eq. (6) or Eq. (8)[6], we propose to minimize the following objective function (Note that $\frac{1}{|C|}$, $\frac{1}{|I_c^+|}$, Z_c are normalizing constants).

$$\underset{g\in\Omega}{\text{argmin}}\sum_{(c,i)\in S}\{1-E\left[\hat{r}(c,i,g)\right]\} \tag{10}$$

where Ω is the set of ranking scoring functions. It is nontrivial to directly optimize $E\left[\hat{r}(c,i,g)\right]$, which is clearly a noncontinuous function. Instead, we propose to minimize an upper bound of Eq. (10) (by leveraging the property $e^{-x}\geq 1-x$ $(x\in\mathbb{R})$) such that it can be fitted into the AdaBoost framework easily.

$$\underset{g\in\Omega}{\text{argmin}}\sum_{(c,i)\in S}\exp\{-E\left[\hat{r}(c,i,g)\right]\} \tag{11}$$

BoostFM is expected to form a strong recommender by linearly combining multiple homogeneous component recommenders[7]. Thus the ranking function (so-called strong recommender) g can be expressed as

$$g^{(t)}=\sum_{t=1}^{T}\beta_t\hat{y}^{(t)} \tag{12}$$

[5] As a 'weak learner', FM fairly meets the basic conditions in both linear complexity and higher prediction accuracy than random guessing.

[6] Maximization of both equations works well in practice, although we only report the results by optimizing Eq. (6) for clarity.

[7] The term weak recommender and component recommender are used interchangeably throughout the paper.

Algorithm 1 BoostFM

Input: The observed context-item interactions S, parameters E and T.
Output: The strong recommender $g^{(T)}$
Initialize $\mathbf{Q}_{ci}^{(t)}=1/|S|$, $g^{(0)}=0$, $\forall(c,i)\in S$
for t =1,..., T **do**
  Create component recommender $\hat{y}^{(t)}$ with $\mathbf{Q}^{(t)}$ on S, $\forall(c,i)\in S$, i.e., Algorithm 2;
  Compute the ranking accuracy $E\left[\hat{r}(c,i,y^{(t)})\right]$, $\forall(c,i)\in S$;
  Compute the coefficient β_t, $\beta_t=\ln\left(\frac{\sum_{(c,i)\in S}\mathbf{Q}_{ci}^{(t)}\{1+E\left[\hat{r}(c,i,y^{(t)})\right]\}}{\sum_{(c,i)\in S}\mathbf{Q}_{ci}^{(t)}\{1-E\left[\hat{r}(c,i,y^{(t)})\right]\}}\right)^{\frac{1}{2}}$;
  Create the strong recommender $g^{(t)}$, $g^{(t)}=\sum_{h=1}^{t}\beta_h\hat{y}^{(h)}$;
  Update weight distribution $\mathbf{Q}^{(t+1)}$, $\mathbf{Q}_{ci}^{(t+1)}=\frac{\exp\{-E\left[\hat{r}(c,i,g^{(t)})\right]\}}{\sum_{(c,i)\in S}\exp\{-E\left[\hat{r}(c,i,g^{(t)})\right]\}}$;
end for

where $\hat{y}^{(t)}$ is the t-th component recommender and $\beta_t\in\mathbb{R}^+$ is the coefficient which is usually determined by the overall recommendation performance of $\hat{y}^{(t)}$ at t-th boosting round. BoostFM runs for T rounds and creates a new component recommender $y^{(t)}$ at each round. Then the newly trained recommender is integrated to the final ensemble recommender $g^{(t)}$. The minimization in Eq. (11) is converted to

$$\underset{\beta_t,y^{(t)}\in\Phi}{\arg\min}\sum_{(c,i)\in S}\exp\{-E\left[\hat{r}(c,i,g^{(t-1)}+\beta_t y^{(t)})\right]\} \tag{13}$$

where Φ is the set of possible component recommenders, and $g^{(t-1)}=\sum_{h=1}^{t-1}\beta_h y^{(h)}$. To solve Eq. (13), we propose to maintain a distribution of weights over each observed (c,i) pair in the training data, denoted by matrix $\mathbf{Q}\in\mathbb{R}^{|C|\times|I|}$. The weight value on the (c,i) training instance at round t is denoted by $\mathbf{Q}_{ci}^{(t)}$. More specifically, the weight distribution reflects the emphasis on the component recommender. At each boosting round, weight values $\mathbf{Q}_{ci}^{(t)}$ on (c,i) pairs with low rank performance by the ensemble strong recommender (i.e., Eq. (12)) are increased so that the component recommender at next boosting round would be forced to give more penalties to those 'hard' training instances. For the implementation of BoostFM, we propose to employ the 'forward stage-wise approach' [14], where $g^{(t)}$ is treated as the additive model, $y^{(t)}$ is the basis function, and β_t is the expansion coefficient of a basis function. BoostFM starts with $g^{(0)}=0$, and then adds new basis functions greedily, without changing the parameters (i.e., Θ) and coefficients of those that have already been added. At each round t, a new expansion coefficient β_t and basis function $y^{(t)}$ can be found to minimize the exponential objective function. More details about the BoostFM have been shown in Algorithm 1. Note that it is computationally expensive to calculate $E\left[\hat{r}(c,i,y^{(t)})\right]$ directly due to the large size of implicit feedback, we solve it by first performing a uniform

sampling to obtain a few non-observed items (say 50), and then calculating the rank of i among them as an unbiased estimator of $\hat{r}(i|c)$. Following [30], it can be proved that there exists a lower bound in terms of the performance measures, as presented in appendix.

Component Recommender

Since this work targets at the top-N recommendation task, we thus propose the ranking optimization methods to create the component recommenders. Naturally, it is feasible to exploit the L2R techniques to optimize Factorization Machines (FM). There are two major approaches in the field of L2R, namely, pairwise and listwise approaches. In the following, we demonstrate ranking factorization machines with both pairwise and listwise optimization.

Weighted Pairwise FM (WPFM)

Inspired by RankNet [3], we employ the weighted cross entropy (CE) as pairwise objective function to learn the preference relations between each two (c,i) pairs. The component recommender based on pairwise optimization is called Weighted Pairwise Factorization Machines (WPFM for short).

Following [3], let $(i,j) \in I$ be a set of item pairs under context $c \in C$, and $\overline{P}^c_{ij}$ be the probability of pairwise relevance ordering from ground truth. For example, let $\overline{P}^c_{ij} \in \{0, 0.5, 1\}$ be defined as 1 if i is more relevant than item j under context c (i.e., $i \succ_c j$), 0 if i is less relevant, and 0.5 if they have the same relevance. The original CE objective function is given

$$L = \sum_{c \in C} \sum_{i \in I} \sum_{j \in I} -\overline{P}^c_{ij} \log P^c_{ij} - \left(1 - \overline{P}^c_{ij}\right) \log\left(1 - P^c_{ij}\right) \tag{14}$$

where P^c_{ij} is the modeled posterior, i.e., $P^c_{ij} = \frac{\exp(\hat{y}(\mathbf{x}^i) - \hat{y}(\mathbf{x}^j))}{1+\exp(\hat{y}(\mathbf{x}^i) - \hat{y}(\mathbf{x}^j))}$. In the implicit feedback settings, only positive observations are available, where non-positive observations are mixed with both non-relevant and unknown items. To simplify Eq. (14), we thus formalize the set of all pairwise relevance relations $D_s \subseteq C \times I \times I$ as

$$D_s = \{(c,i,j) | i \in I^+_c \text{ and } j \in I \backslash I^+_c\}$$

Accordingly, we have $\overline{P}^c_{ij}$=1. Finally, by combining the weight distribution $\mathbf{Q}$, the new objective function and gradient becomes

$$L = -\sum_{(c,,i,j) \in D_s} \mathbf{Q}^{(t)}_{ci} \log \frac{\exp(\hat{y}(\mathbf{x}^i) - \hat{y}(\mathbf{x}^j))}{1 + \exp(\hat{y}(\mathbf{x}^i) - \hat{y}(\mathbf{x}^j))} \tag{15}$$

$$\frac{\partial L}{\partial \theta} = \sum_{(c,,i,j) \in D_s} \lambda_{i,j} \frac{\partial \hat{y}(\mathbf{x}^i) - \partial \hat{y}(\mathbf{x}^j)}{\partial \theta} \tag{16}$$

where $\lambda_{i,j}$ is the learning weight for $(i,j)_c$ pair

$$\lambda_{i,j} = \frac{\partial L((i,j)_c)}{\partial(\hat{y}(\mathbf{x}^i) - \hat{y}(\mathbf{x}^j))} = -\frac{\mathbf{Q}^{(t)}_{ci}}{1 + \exp(\hat{y}(\mathbf{x}^i) - \hat{y}(\mathbf{x}^j))} \tag{17}$$

To deal with the large number of pairs $|D_s|$, an efficient approach is to exploit Stochastic Gradient Descent (SGD) with bootstrap sampling. By differentiating Eq.(1) with respect to θ, we can obtain

$$w_k \leftarrow w_k - \eta(\lambda_{i,j}(x^i_k - x^j_k) + \gamma_{w_k} w_k) \tag{18}$$

$$v_{k,f} \leftarrow v_{k,f} - \eta(\lambda_{i,j} G(x^i_k, x^j_k) + \gamma_{v_{k,f}} v_{k,f}) \tag{19}$$

Algorithm 2 Component Recommender Learning

Input: The set of all pairwise relations D_s, the weight distribution $\mathbf{Q}^{(t)}$, hyper-parameters γ_θ, η
Output: $\Theta = (\mathbf{w}, \mathbf{V})$
Initialize Θ: $\mathbf{w} \leftarrow (0,...,0)$; $\mathbf{V} \sim \mathcal{N}(0, 0.1)$;
for $e = 1, ..., maxiter$ **do**
  Uniformly Draw (c,i) from S ;
  Uniformly Draw j from $I \backslash I^+_c$;
  for $f \in \{1,...,d\}$ **do**
    for $k \in \{1,...,n\} \wedge x_k \neq 0$ **do**
      Update $v_{k,f}$ as in Eq. (19);
    end for
  end for
  for $k \in \{1,...,n\} \wedge x_k \neq 0$ **do**
    Update w_k as in Eq. (18);
  end for
end for

where

$$G(x^i_k, x^j_k) = \sum_{l=1}^{n} v_{l,f}(x^i_k x^i_l - x^j_k x^j_l) - v_{k,f}({x^i_k}^2 - {x^j_k}^2) \tag{20}$$

γ_θ (i.e., γ_{w_k}, $\gamma_{v_{k,f}}$) is a hyper-parameter for the L2 regularization, and η is the learning rate. Algorithm 2 shows how the component recommender is optimized with weighted pairwise constraints.

Weighted 'Listwise' FM (WLFM)

For the top-N item ranking task, the recommendation quality is highly position-dependent, i.e., the accuracy of items near the top of the ranked list is more important to that of the low-position items [31]. In this sense, the pairwise objective function might be a suboptimal scheme for item recommendation tasks, although it gains much popularity according to recent literature. Instead, listwise approaches solve this problem in a more elegant way where the models are formalized to directly optimize a specific list-level ranking metric, such as NDCG. However, it is difficult to directly optimize the ranking metrics because they are either non-continuous or non-differentiable. To solve this challenge, our method bypasses the problem by adding listwise information into WPFM, the way of which is referred to as Weighted 'Listwise' Factorization Machines (WLFM[8]). The implementation of WLFM is inspired by the design idea of LambdaRank [19].

Following [19], we design a similar lambda function as $f(\lambda_{i,j}, \zeta_c)$, where ζ_c is the current item ranking list for context c. For example, we use NDCG as the ranking measure, $f(\lambda_{i,j}, \zeta_c)$ is given

$$f(\lambda_{i,j}, \zeta_c) = \lambda_{i,j} |\triangle NDCG(c)_{ij}| \tag{21}$$

where $\triangle NDCG(c)_{ij}$ is the size of NDCG change for c when the positions of items i, j get swapped, computed by

$$\triangle NDCG(c)_{ij} = Z_c\left(\frac{1}{\log_2(\hat{r}(i|c)+1)} - \frac{1}{\log_2(\hat{r}(j|c)+1)}\right) \tag{22}$$

[8] The general formulation of WLFM is listwise, although its implementation in practice is pairwise. Please also note that in [31] LFM is treated as a single model while LFM in this paper serves as component recommenders with non-uniform boosting weights.

That is, WLFM can be implemented by replacing $\lambda_{i,j}$ with $f(\lambda_{i,j}, \zeta_c)$ in Eqs. (18) and (19). The implementation is reasonable for Information Retrieval (IR) tasks but unfortunately infeasible for recommendation tasks based on implicit feedback. The reason is to calculate $\triangle NDCG(c)_{ij}$ of different item pairs, the recommender has to score all the items in ζ_c to find the rank of i and j. In IR tasks, the candidate documents returned by the retrieval model have already been reduced to a small size [34]. However, in recommendation scenario, since there is no query to filter candidate items, all unobserved items have to be considered as candidates (i.e., $|\zeta_c| = |I|$). That means the additional computational complexity before each parameter update is $O(|I| \cdot T_{pred})$, where T_{pred} is the run time to predict an item score by FM. The huge computational complexity is obviously infeasible in practice.

To solve the efficiency problem, we propose a lambda-motivated sampling scheme following [31]. Suppose an ideal lambda function $f(\lambda_{i,j}, \zeta_c)$ for each training pair $(i,j)_c$ is given with the current item ranking list ζ_c, the idea is that if we have a scheme that generates the training item pairs proportional to the probability $f(\lambda_{i,j}, \zeta_c)/\lambda_{i,j}$ (just like $\triangle NDCG(c)_{ij}$), then we can achieve an almost equivalent training effect. That means we should draw item pairs according to probability distribution $p(j|c) \propto f(\lambda_{i,j}, \zeta_c)/\lambda_{i,j}$, i.e., drawing more item pairs if they generate a larger $\triangle NDCG$ after swapping. To illustrate which item pair would generate a larger $\triangle NDCG$ by swapping, we give a schematic of a ranked list as below, where $+1$ and -1 are observed and non-observed items respectively.

$$\text{Rank Order}: \overbrace{-1,-1,+1,-1,\ \underbrace{-1,-1,+1}_{\triangle NDCG(c)_{75}=0.033}}^{\triangle NDCG(c)_{71}=0.409}, -1, ..., -1$$

We observe that the value of $\triangle NDCG(c)_{71}$ is larger than that of $\triangle NDCG(c)_{75}$. This implies $\triangle NDCG(c)_{ij}$ is likely to be larger if non-observed items have a higher rank. This is because the high ranked non-observed items hurt the ranking performance more than the low ranked ones. However, we still have the same computational complexity issue since we need to score all non-observed items before finding one with a higher rank. To overcome the challenge, we exploit an item popularity sampling strategy by drawing more non-observed items with high popularity as substitutes for high ranked non-observed items. This is because an ideal item ranker is supposed to assign higher ranks to observed items than non-observed ones. As we know, popular items have more chances to be an observed item. Hence, it is reasonable to devise a popularity-aware sampler to replace the uniform one in Algorithm 2. We implement a sampler p_j by sampling popular items proportionally to the empirical item popularity distribution[9], e.g., exponential or power-law distribution. p_j is given below

$$p_j \propto \exp(-\frac{r(j)}{|I| \times \rho}), \rho \in (0,1] \tag{23}$$

where $r(j)$ represents the rank of item j among all items I according to the overall popularity, ρ is a tuning parameter for the distribution. Therefore, for WLFM, Line 6 in Algorithm 2 can be replaced by the above sampler. Regarding the complexity, the popularity-aware sampler does not increase the complexity because the popularity distribution is static and can thus be calculated in advance.

[9] In practice, the item popularity distribution in recommender datasets usually follows approximately a long-tail distribution [31, 22].

Table 1: **Basic statistics of datasets. Each tuple represents an observed context-item interaction. Note that tags in MLHt dataset are regarded as recommended items (i.e., j in $\mathbf{x}_j$), while a user-item (i.e., user-movie) pair is regarded as context (i.e., c in $\mathbf{x}_c$).**

Datasets	Users	Items	Tags	Artists	Albums	Tuples
MLHt	2113	5908	9079	-	-	47958
Last.fm	983	60000	-	25147	-	246853
Yahoo	2450	124346	-	9040	19851	911466

EXPERIMENTS

In this section, we conduct comprehensive experiments on three publicly accessible datasets to verify the effectiveness of BoostFM in various settings.

Experimental Setup

Datasets, Evaluation Metrics and Baselines

We use three CF datasets for our experiments, namely, MovieLens Hetrec (MLHt)[10] (user-movie-tag triples, where the context is a user-movie pair, the item is the tag), Last.fm[11] (user-music-artist triples, where the context is the user, the item is a music track with an artist) and Yahoo music[12] (user-music-artist-album tuples, where the context is the user, the item is a music track with an artist and album). In the MLHt dataset, the task is to recommend top-N relevant tags for each user-movie pair, while in Last.fm and Yahoo datasets, it is to recommend top-N preferred music tracks (with item side information) to each user. To speed up the experiments, we follow the common practice as in [6, 31] by randomly sampling a subset of users from the user pool of the Yahoo dataset, and a subset of items from the item pool of the Last.fm dataset. The MLHt dataset is kept in its original form. The statistics of the datasets after preprocessing are summarized in Table 1.

To evaluate the top-N recommendation quality of BoostFM, we present the results with two standard ranking metrics used in previous recommendation literature [14, 17], namely, Precision@N and Recall@N (denoted by Pre@N and Rec@N respectively)[13], where N is the number of recommended items (again, tags are considered as items in the MLHt datasets).

For top-N recommendation, with given context c, we compute Pre@N and Rec@N as follows [14]:

$$\text{Pre@N(c)} = \frac{tp_c}{tp_c + fp_c} \quad \text{and} \quad \text{Rec@N(c)} = \frac{tp_c}{tp_c + tn_c} \tag{24}$$

where tp_c is the number of items contained in both the ground truth and the top-N rank list predicted by algorithms; fp_c is

[10] http://grouplens.org/datasets/hetrec-2011/

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

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

[13] Please note that we have verified our algorithm with other ranking metrics (i.e., MAP, NDCG, MRR), all of which show a consistent trend in performance, but are omitted here for saving space.

the number of items contained in the top-N rank list but not in the ground truth; and tn_c is the number of items contained in the ground truth but not in the top-N rank list. The final performance reported is an average of Pre@N (Rec@N) values of all context.

In our experiments, we compare our BoostFM with several powerful CF baselines. For clarity, we refer to BoostFM with WPFM and WLFM as B.WPFM and B.WLFM respectively. For tag recommendation in the MLHt dataset, we utilize Most Popular (MP) [17], Factorization Machines (FM) [21] and Pairwise Interaction Tensor Factorization (PITF)[14] [24] as baselines. For music recommendation based on item side information, we utilize MP, User-based Collaborative Filtering (UCF)[15] [33], FM, Bayesian Personalized Ranking (BPR) [23], and Pairwise Ranking Factorization Machines (PRFM) by applying the CE loss and Hinge loss [18] (denoted by PRFM.CE and PRFM.H respectively).

Hyper-parameter Settings

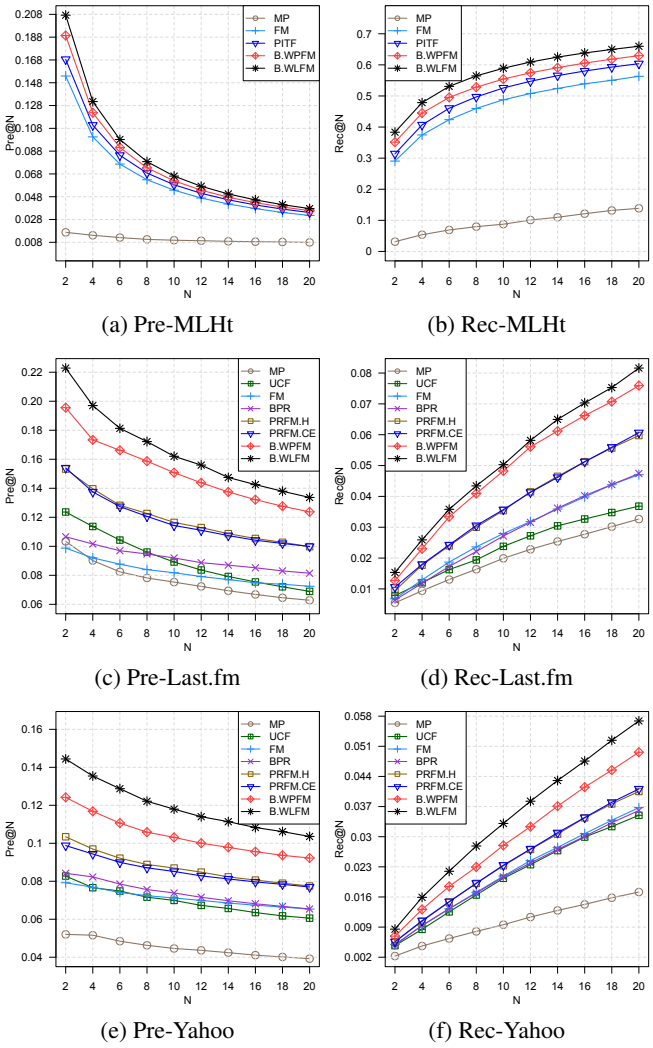
When performing the learning algorithms, there are several critical hyper-parameters needed to be set. (1) The number of component recommender T: For the purpose of comparison, T of BoostFM is set to 10 in all three datasets if not explicitly declared. The contribution of T will be discussed later. (2) Learning rate η and regularization γ_θ: We first employ the 5-fold cross validation to find the best η by running BoostFM with $\eta \in \{0.005, 0.01, 0.02, 0.05, 0.08, 0.1, 0.2, 0.4\}$, and then tune γ_θ in the same way while fixing η. Specifically, η is set to 0.08 in the Last.fm and Yahoo datasets, and 0.4 in the MLHt dataset; γ_θ is set to 0.05, 0.02 and 0.005 in the Last.fm, Yahoo and MLHt dataset respectively[16]. In our experiments, we find all FM based models perform well enough by just employing polynomial term (see Eq. (1)), and thus we omit the configuration of the linear term. Baseline algorithms are tuned in the same way. (3) Latent dimension d: For comparison purposes, it is common practice (e.g., [17, 33]) to assign a fixed d value (e.g., $d = 30$ in all experiments) for all methods based on factorization models. Results for d=10, 50, 100 show similar behavior but are omitted for space reasons. (4) Distribution coefficient ρ: $\rho \in (0, 1]$ is specific for the WLFM component recommender, which is tuned according to the popularity distribution of recommender datasets.

Performance Evaluation

All experiments are conducted with the standard 5-fold cross validation. The average results over 5 folds are reported as the final performance.

Accuracy Summary

Figure 1(a-f) shows the prediction quality of all algorithms in the three datasets. Several insightful observations can be made: First, all personalized[17] algorithms (UCF, BPR, PITF, FM, PRFM) generally outperform MP, which is a non-personalized



(a) Pre-MLHt (b) Rec-MLHt

(c) Pre-Last.fm (d) Rec-Last.fm

(e) Pre-Yahoo (f) Rec-Yahoo

Figure 1: **Performance comparison w.r.t. top-N values, i.e., Pre@N & Rec@N. N ranges from 2 to 20, the number of component recommender T is fixed to 10, and ρ for B.WLFM is fixed to 0.3.**

method. Besides, our BoostFM (i.e., B.WPFM and B.WLFM) clearly outperforms all the counterparts in terms of two standard top-N recommendation metrics. Second, by setting a larger N, values of Pre@N get lower and values of Rec@N become higher. This observation reveals the typical behavior of recommender systems: the more items are recommended, the better the recall but the worse the precision is achieved [31].

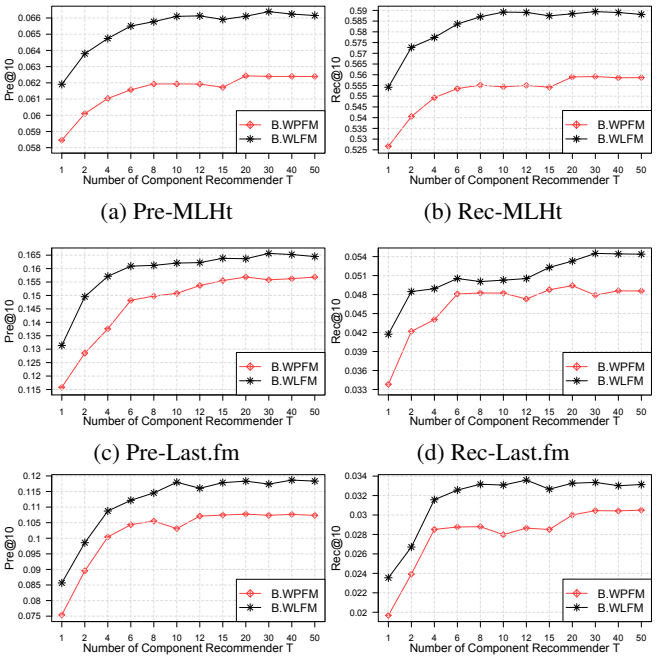
Regarding the effectiveness of factorization models, we make the following observations: (1) Although FM incorporates additional side information, it still performs poorer than BPR (without side information) in Last.fm and Yahoo datasets, which suggests that pairwise approaches may achieve better results than pointwise approaches for item ranking tasks. This is because the only difference (except from the prediction functions) between FM and BPR is that they utilize different loss functions, i.e., quadratic and negative log-likelihood

[14] We use a shared tag latent factor for the interaction between users and items.

[15] Pearson correlation is used in this work to compute user similarity and the top-20 most similar users are selected as the nearest neighbors.

[16] Note FM based models have several regularization parameters, including γ_{w_k}, $\gamma_{v_{k,f}}$. We borrow the idea from [21] by grouping them for each factor layer, i.e., $\gamma_\pi = \gamma_{w_k}$, $\gamma_\xi = \gamma_{v_{k,f}}$.

[17] For factorization models, personalization means each context should attain one set of parameters.

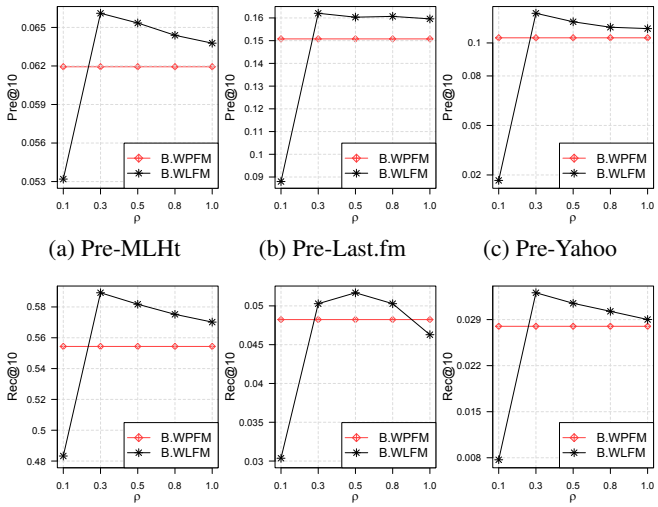


(a) Pre-MLHt (b) Rec-MLHt

(c) Pre-Last.fm (d) Rec-Last.fm

(e) Pre-Yahoo (f) Rec-Yahoo

Figure 2: **Performance trend of BoostFM (i.e., B.WPFM and B.WLFM) by tuning T w.r.t. Pre@10 & Rec@10. T ranges from 1 to 50, and ρ is fixed to 0.3.**



(a) Pre-MLHt (b) Pre-Last.fm (c) Pre-Yahoo

(d) Rec-MLHt (e) Rec-Last.fm (f) Rec-Yahoo

Figure 3: **Performance trend of BoostFM by tuning ρ w.r.t. Pre@10 & Rec@10. $\rho \in \{0.1, 0.3, 0.5, 0.8, 1.0\}$, $T = 10$.**

loss respectively. (2) PRFM (i.e., PRFM.CE and PRFM.H) largely outperforms BPR, which clearly shows the effectiveness of recommendation with side information (e.g., artists and albums). (3) Among all the baseline algorithms, the performance of PRFM and PITF[18] is very promising. We believe two main factors contribute to the appealing results. First, by applying pairwise ranking losses PRFM and PITF are more appropriate to handle top-N item recommendation tasks than FM (pointwise); Second, by applying FM and tensor factorization as ranking functions, PRFM and PITF are more effective to predict preference orderings (by context and side information) than BPR.

BoostFM vs. PRFM and PITF We observe in Figure 1 that our BoostFM (i.e., B.WPFM and B.WLFM) consistently outperforms the state-of-the-art methods PITF and PRFM, measured by the two top-N ranking metrics. For example, in the MLHt dataset, we can calculate that B.WPFM outperforms PITF by 6.1% and 5.4% in terms of Pre@10 and Rec@10 respectively[19]. In particular, the significant improvements by B.WPFM (compared with PRFM.CE and PRFM.H) are more than 18% on Pre@10 and 35% on Rec@10 in both the Last.fm and Yahoo datasets. The results shows that the accuracy of top-N recommendation can be significantly improved by using the boosting technique.

[18] PRFM is identical to PITF (in terms of ranking functions) when $\mathbf{x}$ is composed of three non-zero categorical variables.

[19] To save space, we only use the top-10 (i.e., Pre@10 and Rec@10) value for the following descriptions since the performance trend on other top-N values is consistent.

B.WPFM vs. B.WLFM In contrast to B.WPFM, B.WLFM achieves much better results consistently across all datasets for ranking metrics in Figure 1. The difference is that the component recommender WLFM is trained by optimizing a ranking measure while WPFM is trained by minimizing pairwise loss which is position-independant. The results clearly validate our previous analysis and demonstrate the effectiveness of our proposed 'listwise' strategy.

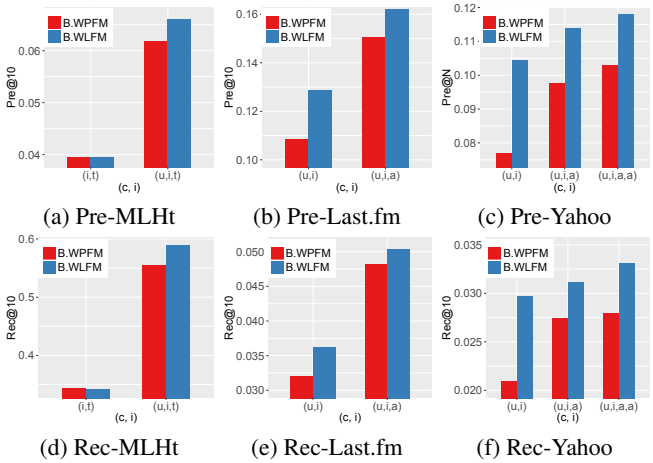
Effect of Number of Component Recommenders

In this subsection, we evaluate the performance sensitivity of BoostFM to the number of component recommenders T. T is adjusted from 1 to 50 and ρ for B.WLFM is set to a fixed value[20] (i.e., 0.3) for a fair comparison. The results of all datasets are summarized in Figure 2, in terms of Pre@10 and Rec@10. We observe that the top-10 recommendation performance generally improves by increasing the number of component recommenders T, particularly when T is smaller than 10. When T is larger than 10, adding more component recommenders generates marginal performance improvements. In addition, we can observe that B.WLFM performs significantly better than B.WPFM, which is consistent with previous results in Figure 1. Note that when T is set to 1, B.WPFM reduces to PRFM.CE. The different performance between them comes from their parameter settings since the best hyper-parameters of B.WPFM are found based on $T = 10$.

Effect of Parameter ρ

In this subsection, we study the effect of the second component recommender WLFM with the parameter ρ. By tuning $\rho \in \{0.1, 0.3, 0.5, 0.8, 1.0\}$, we depict the results in Figure 3. First, we clearly see that BoostFM with WLFM performs much better than that with WPFM when $\rho \in [0.3, 1.0]$. Particularly, WLFM produces the best accuracy when setting ρ to

[20] Again, the performance trend keeps consistent for any value of $\rho \in \{0.1, 0.3, 0.5, 0.8, 1.0\}$.



(a) Pre-MLHt (b) Pre-Last.fm (c) Pre-Yahoo

(d) Rec-MLHt (e) Rec-Last.fm (f) Rec-Yahoo

Figure 4: **Performance comparison w.r.t. Pre@10 & Rec@10 with context and side information. In (a) (d), (i,t) denotes an item-tag (i.e., movie-tag) pair and (u,i,t) denotes a user-item-tag triple; in (b)(c)(e)(f) (u,i) denotes a user-item (i.e., user-music) pair and (u,i,a) denotes a user-item-artist triple; similarly, (u,i,a,a) denotes a user-item-artist-album quad. T is fixed to 10, and ρ is fixed to 0.3.**

0.3 across nearly all datasets, but then the performance experiences a significant decrease when setting it to 0.1. The reason is because the component recommender WLFM concentrates more gradient steps on the most popular items due to the over-sampling scheme when ρ is set to a small value (i.e., $\rho = 0.1$) based on Eq. (23). In this case, most less unpopular items will not be chosen for training, and thus the model is under-trained. In practice, we would suggest to tune the parameter ρ gradually from a larger value (e.g., 1.0) to a smaller one and find the one that performs the best in the training set. Empirically, a relatively smaller ρ can be set on a recommendation dataset with a longer tail.

Effect of Adding Features

Finding competitive context and auxiliary features is not the main focus of our work but it is interesting to see to how BoostFM improves the performance by adding feature information. Therefore, we conduct a contrast experimentation and show the results in Figure 4. First, we observe BoostFM performs much better with (u,i,a) tuples than that with (u,i) tuples in Last.fm and Yahoo datasets. This result is intuitive as a user may like a music track if she likes another one by the same artist. Second, as expected, BoostFM with (u,i,a,a) tuples performs further better than that with (u,i,a) tuples in (c) and (f). The results verify the effectiveness of BoostFM in modeling item attribute information. In addition, similar result can be observed in the MHLt dataset: BoostFM achieves largely better results with (u,i,t) tuples than that with (i,t) tuples. We draw the conclusion that the more effective feature information incorporated, the better BoostFM performs. Interestingly, we find an outlier result in the MHLt dataset where the B.WLFM does not outperform B.WPFM with (i,t) pair. By manual inspection, we find that the tags for each user-movie (i.e., c) pair is highly dominated by the user's preference (e.g., tagging habits) rather than the movie itself. In other words, some users assign the same tags for many different movies. The user bias problem can be well handled when the user information is considered during the pairwise comparison (e.g, BoostFM with (u,i,t) tuples), while B.WLFM usually leads to non-improved results without the user context. The reason might be that the overall popular tags for movies may not be popular tags for user-movie pairs.

CONCLUSION AND FUTURE WORK

In this paper, we have proposed a novel ranking predictor Boosted Factorization Machines (BoostFM) for top-N recommendation with the incorporation of rich feature information. Inheriting advantages from both boosting technique and FM, BoostFM is (i) capable of improving top-N recommendation performance by forming an ensemble of component recommenders; (ii) very flexible to integrate auxiliary information, such as context and item side information, by mapping them into a general feature space. Regarding the optimization of component recommenders, we have devised both pairwise and 'listwise' version ranking FM with an adaptive re-weighting scheme. Extensive results on three public datasets have shown that the suggested BoostFM (i.e., B.WPFM and B.WLFM) clearly outperformed a bunch of state-of-the-art CF counterparts.

The proposed BoostFM can be applied to other ranking domains based on implicit feedback and sparse feature information, such as personalized collaborative retrieval and learning to personalize query auto-completion. For future work, we intend to investigate its performance in these domains.

ACKNOWLEDGMENTS

Fajie thanks the CSC funding for supporting his research. This work was also partially supported by the National Natural Science Foundation of China (No. 61472073).

REFERENCES

1. Linas Baltrunas and Francesco Ricci. 2009. Context-based splitting of item ratings in collaborative filtering. In *RecSys*. 245–248.
2. Alberto Bertoni, Paola Campadelli, and M Parodi. 1997. A boosting algorithm for regression. In *ICANN*. 343–348.
3. Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In *ICML*. 89–96.
4. Chen Cheng, Fen Xia, Tong Zhang, Irwin King, and Michael R Lyu. 2014. Gradient boosting factorization machines. In *RecSys*. 265–272.
5. Nipa Chowdhury, Xiongcai Cai, and Cheng Luo. 2015. BoostMF: Boosted Matrix Factorisation for Collaborative Ranking. In *ECML-PKDD*. 3–18.
6. Konstantina Christakopoulou and Arindam Banerjee. 2015. Collaborative Ranking with a Push at the Top. In *WWW*. 205–215.
7. Yoav Freund and Robert Schapire. 1999. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence* (1999), 1612.

8. Yoav Freund and Robert E Schapire. 1995. A desicion-theoretic generalization of on-line learning and an application to boosting. In *EuroCOLT*. Springer, 23–37.

9. Jerome Friedman, Trevor Hastie, Robert Tibshirani, and others. 2000. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics* (2000), 337–407.

10. Liangjie Hong, Aziz S Doumith, and Brian D Davison. 2013. Co-factorization machines: modeling user interests and predicting individual decisions in twitter. In *WSDM*. 557–566.

11. Xiaotian Jiang, Zhendong Niu, Jiamin Guo, Ghulam Mustafa, Zi-Han Lin, Baomi Chen, and Qian Zhou. 2013. Novel Boosting Frameworks to Improve the Performance of Collaborative Filtering. (2013).

12. Alexandros Karatzoglou, Xavier Amatriain, Linas Baltrunas, and Nuria Oliver. 2010. Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *RecSys*.

13. Yehuda Koren. 2010. Collaborative filtering with temporal dynamics. *Commun. ACM* (2010), 89–97.

14. Xutao Li, Gao Cong, Xiao-Li Li, Tuan-Anh Nguyen Pham, and Shonali Krishnaswamy. 2015. Rank-GeoFM: a ranking based geographical factorization method for point of interest recommendation. In *SIGIR*. 433–442.

15. Xuchun Li, Lei Wang, and Eric Sung. 2008. AdaBoost with SVM-based component classifiers. *EAAI* (2008), 785–795.

16. Yong Liu, Peilin Zhao, Aixin Sun, and Chunyan Miao. 2015. A boosting algorithm for item recommendation with implicit feedback. In *IJCAI*.

17. W. Pan and L. Chen. 2013. GBPR: Group preference based bayesian personalized ranking for one-class collaborative filtering. In *IJCAI*. 2691–2697.

18. Runwei Qiang, Feng Liang, and Jianwu Yang. 2013. Exploiting ranking factorization machines for microblog retrieval. In *CIKM*. 1783–1788.

19. C Quoc and Viet Le. 2007. Learning to rank with nonsmooth cost functions. (2007), 193–200.

20. Steffen Rendle. 2010. Factorization machines. In *ICDM*. 995–1000.

21. Steffen Rendle. 2012. Factorization Machines with libFM. *TIST* (2012), 57.

22. Steffen Rendle and Christoph Freudenthaler. 2014. Improving pairwise learning for item recommendation from implicit feedback. In *WSDM*. 273–282.

23. Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *UAI*. 452–461.

24. Steffen Rendle and Lars Schmidt-Thieme. 2010. Pairwise Interaction Tensor Factorization for Personalized Tag Recommendation. In *WSDM*. 81–90.

25. Robert E Schapire. 2013. Explaining adaboost. In *Empirical inference*. 37–52.

26. Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, and Alan Hanjalic. 2014. Cars2: Learning context-aware representations for context-aware recommendations. In *CIKM*. 291–300.

27. Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, Alan Hanjalic, and Nuria Oliver. 2012. TFMAP: Optimizing MAP for top-n context-aware recommendation. In *SIGIR*. 155–164.

28. Yanghao Wang, Hailong Sun, and Richong Zhang. 2014. Adamf: Adaptive boosting matrix factorization for recommender system. In *WAIM*. 43–54.

29. Liang Xiong, Xi Chen, Tzu-Kuo Huang, Jeff G Schneider, and Jaime G Carbonell. Temporal Collaborative Filtering with Bayesian Probabilistic Tensor Factorization. SIAM.

30. Jun Xu and Hang Li. 2007. Adarank: a boosting algorithm for information retrieval. In *SIGIR*. 391–398.

31. Fajie Yuan, Guibing Guo, Joemon M Jose, Long Chen, Haitao Yu, and Weinan Zhang. 2016a. Lambdafm: learning optimal ranking with factorization machines using lambda surrogates. In *CIKM*. 227–236.

32. Fajie Yuan, Guibing Guo, Joemon M Jose, Long Chen, Haitao Yu, and Weinan Zhang. 2016b. Optimizing factorization machines for top-n context-aware recommendations. In *WISE*. 278–293.

33. Fajie Yuan, Joemon M Jose, Guibing Guo, Long Chen, Haitao Yu, and Rami S Alkhawaldeh. 2016c. Joint Geo-Spatial Preference and Pairwise Ranking for Point-of-Interest Recommendation. In *ICTAI*.

34. Weinan Zhang, Tianqi Chen, Jun Wang, and Yong Yu. 2013. Optimizing top-n collaborative filtering via dynamic negative item sampling. In *SIGIR*. 785–788.

APPENDIX

Theorem 1 indicates that the ranking accuracy in terms of the performance measures can be continuously grown.

THEOREM 1. *The following bound holds in terms of ranking accuracy (e.g., AUC) of the BoostFM algorithm on the training data:*

$$\frac{1}{|C|}\sum_{(c,i)\in S}\frac{1}{|I_c^+|}E\left[\hat{r}(c,i,g)\right] \geq \frac{\sum_{(c,i)\in S}\frac{1}{|I_c^+|}}{|C|}\left[1-\prod_{t=1}^{T}e^{-\delta_{min}^t\sqrt{1-\pi(t)^2}}\right]$$

where $\pi(t)=\sum_{(c,i)\in S}\boldsymbol{Q}_{ci}^{(t)}E\left[\hat{r}(c,i,y^{(t)})\right]$, $\delta_{min}^t=min_{(c,i)\in S}\delta_{ci}^t$, *and* $\delta_{ci}^t = E\left[\hat{r}(c,i,g^{(t-1)}+\beta_t y^{(t)})\right] - E\left[\hat{r}(c,i,g^{(t-1)})\right] - \beta_t E\left[\hat{r}(c,i,y^{(t)})\right]$, *for all* $(c,i)\in S$ *and* $t=1,2,...,T$.