BoostMF: Boosted Matrix Factorisation for Collaborative Ranking

Nipa Chowdhury(✉), Xiongcai Cai, and Cheng Luo

The University of New South Wales, Sydney, NSW 2052, Australia
{nipac,xcai,luoc}@cse.unsw.edu.au

Abstract. Personalised recommender systems are widely used information filtering for information retrieval, where matrix factorisation (MF) has become popular as a model-based approach to personalised recommendation. Classical MF methods, which directly approximate low rank factor matrices by minimising some rating prediction criteria, do not achieve a satisfiable performance for the task of top-N recommendation. In this paper, we propose a novel MF method, namely BoostMF, that formulates factorisation as a learning problem and integrates boosting into factorisation. Rather than using boosting as a wrapper, BoostMF directly learns latent factors that are optimised toward the top-N recommendation. The proposed method is evaluated against a set of state-of-the-art methods on three popular public benchmark datasets. The experimental results demonstrate that the proposed method achieves significant improvement over these baseline methods for the task of top-N recommendation.

Keywords: Recommender system · Collaborative filtering · Matrix factorisation · Learning to rank · Boosting

1 Introduction

Recommender systems (RS) have gained much attention in information retrieval (IR) to guide users when searching information from the information pool. Collaborative filtering (CF) is widely used to build personalised recommender systems such as book recommendation in Amazon [2], movie recommendation in Netflix [2] and friend recommendation in Facebook [2]. It aims to predict the preference of a user on its unseen items by learning the preference from the historic feedback of this user and other like-minded users to provide the user with a list of recommended items or prediction score of items. The personalised prediction problem [1–3,15] in presenting recommendation list can be regarded as estimating the preference function in CF. Usually, this problem can be solved by either i) generating the recommendation list by sorting the predicted ratings in descending order, known as rating-oriented CF or ii) learning the ranking function directly, known as ranking-oriented CF. When the recommendation list itself becomes large, it will be obsolete since people prefer only top listed items

© Springer International Publishing Switzerland 2015

DOI: 10.1007/978-3-319-23525-7_1

[2,15]. So recommender systems should not only be optimised to reflect user tastes and preferences but also rank top items correctly.

Matrix factorisation is a popular model-based CF method, which demonstrates great success in Netflix prize competition [7]. In MF, given N users and M items, the user-item preference matrix $R \in \Re^{N \times M}$ can be approximated by two low rank matrices $P \in \Re^{N \times K}$ and $Q \in \Re^{M \times K}$ as $R \approx P \cdot Q^{'}$ by minimising the sum of squared errors, where $K \ll min(N, M)$ is the dimensionality of latent factors representing user preferences and item characteristics. The major purpose of MF is to obtain some forms of lower-rank approximation to original matrix for understanding the interaction of user preferences and item attractiveness in forms of latent factors [7].

Nevertheless, traditional matrix factorisation algorithms [7,13] based on rating-oriented CF do not achieve satisfactory ranking performance in the task of top-N recommendation [2,14,15,17]. As users are more concerned about recommended items in the top of the recommendation list, items with higher ratings (i.e., higher possibilities to be preferred by users) should be modelled more correctly than low rating items. Hence, it is important to consider the accuracy of ranked list during learning, and give different emphasises on items with different users' feedback. However, the conventional approach usually does not discriminate the significances of different feedback of items, and the learned latent factors representing user preferences and item characteristics are thus not the optimal ones for generating personalised recommendations. Meanwhile, most of existing methods assume that each latent factor could not contribute differently during the learning of user preferences and item characteristics. This assumption leads to simply update the latent factors as a whole, which may not perform well. In reality, users who originate from different backgrounds are highly proportional to select preferable items based on their different characteristics. These various characteristics are compactly represented by different latent factors. Furthermore, most of existing methods for the top-N recommendation task minimise some error metrics, such as the sum of squared errors, to generate the recommendation list. Unlike optimising against some ranking metrics such as the one used in the paper, this approach is actually an indirect approach that degrades the ranking performance. For example, probabilistic matrix factorisation [13] (PMF), which forms the basis of many model-based recommendation algorithms, adopts even weights on all items and learns all latent factors at a time by minimising the sum of squared errors via stochastic gradient descent.

To improve the accuracy of the top ranked items in the recommendation lists during learning and exploit the contribution of each latent factor separately, we develop a novel method, namely BoostMF, that uses boosting to learn the low rank factor matrices by directly optimising the ranking measure to improve top-N recommendation performance. Specially, rather than treating all observed items with equal importance for each user, BoostMF imposes different emphasises on observed items using a personalised feature selection scheme based on the current estimation of IR evaluation measure. Without computing any structured estimation of ranking loss or continuous approximations of non-smooth IR measure, the proposed method optimises the IR measure directly by integrating boosting into the optimisation, i.e. by gradient descent, of matrix factorisation methods. As the iteration of the optimisation procedure continues, the algorithm is able to place more focus on training examples that have not yet been ranked in top positions correctly. As in real-world deployment, users are more interested in top-N recommended items; this shifting on focus is important and rational for our method to achieve an improved recommendation performance, which will be demonstrated in Section 4. In the end, the learned latent factors representing user preferences and item characteristics are more suitable for generating top-N recommendation. To empirically study the performance of BoostMF, we evaluate our algorithm with some state-of-the-art methods in top-N recommendation and the results demonstrate that our method significantly outperforms these methods for top-N recommendation in terms of recommendation accuracy. Because contextual information is sensitive and expensive to collect, we only focus on user feedback without bothering contextual information. Therefore, we do not compare our method with other rating or ranking-oriented CF methods that use contextual information in addition to user feedback.

The rest of the paper is arranged as follows: in Section 2, we summarise related work and place our work with respect to it. In Section 3, we present the proposed boosted matrix factorisation method. Experimental results are presented in Section 4. Finally, we draw conclusions in Section 5.

2 Related Work

Learning to rank (LTR) is an important research direction in information retrieval where the goal is to present a ranked list of information in response to a query or request [10]. AdaRank [18], MPBoost [19], and RankBoost [4] are well known LTR methods that use boosting to improve ranking performance. If we consider a query as a user and a list of information as items, recommender systems focus on the personalised view of same ranking task as that of LTR. However, incorporating LTR techniques in personalised recommendation is challenging. LTR methods can only handle non-personalised ranking problems rather than personalised ranking and recommendation problems, and also consider that feature vector of items are given and unchanged during learning. But in recommendation settings, user feature and item feature are not explicitly presented during training. The challenge also arises from learning the low rank matrices by optimising the training criterion which is different from the final evaluation criterion that is used to measure the ranking performance. Although different approaches [2,14,15,17] in LTR are adopted to minimise the ambiguity between learning objective criterion and final evaluation measure, these methods either have unsatisfactory performance or incur with computational overhead. The developed BoostMF method in this paper thus aims to simultaneously learn feature vectors and optimise ranking.

Existing methods in ranking-oriented CF can be generally divided into three categories based on the type of issues needed to be addressed. The first class of methods relies on the transformation of ranking measure. CofiRank [17] and CLiMF [14] are the methods that fall into this category. CofiRank uses structured estimation of the ranking loss and CLiMF derives a lower bound of the smooth ranking measure to solve ranking problem in recommendation. However, these transformation results to significantly computational overhead. Our proposed BoostMF algorithm directly accounts the final evaluation criteria into approximating low rank matrices from a high rank matrix. Specifically, based on PMF, we incorporate boosting procedure to learn low rank factor matrices directly for top-N recommendations. To the best of our knowledge, this approach has not been applied before for top-N recommendations. The second class of methods views the recommendation problem as a list-wise ranking problem and uses list-wise loss functions. For example, ListRank-MF [15] uses list-wise loss function based on cross entropy of the top one probability of items. Unified recommendation model (URM) [16] combines both rating-oriented CF, (i.e., PMF) and a ranking-oriented CF, (i.e., ListRank-MF) to improve ranking performance. However, these methods optimise loss function which is not directly related to the final ranking measure, which is not optimal to improve the performance of top-N recommendation. In this regard, BoostMF employs personalised weak ranker at each round to relate the final evaluation measure into the learning process of the model. The third class of methods solves the ranking problem as a regression problem. In collaborative ranking [2], PMF is used to generate feature vectors and regression based LTR algorithm (i.e., point-wise and pair-wise) is constructed by these feature vectors to produce the ranking. OrdRec is proposed in [8] as a CF framework following point based approach, and it aims to minimise ordinal regression loss. BPR-MF uses [12] different pair-wise optimisation criterion where pairs are formed by taking one from observed items and the other from unobserved items by assuming a user prefers observed items over unobserved items. But, these methods optimise ranking criterion which is different from the final evaluation measure and hence the final ranking measure is not directly applied to the learning process of the model. The learning model of these methods also imposes equal errors on items misplacement in all positions of the recommendation list. However, in BoostMF, the final ranking measure is directly related to the learning process of the model. BoostMF also uses personalised weight distribution for each user on its rated items to emphasize errors of the learning model on misplacing items in higher positions than lower positions. Thus BoostMF is able to generate better recommendation list which is optimal for the task of top-N recommendation.

3 Boosted Matrix Factorisation (BoostMF)

In this section, we firstly present a key component related to our algorithm probabilistic matrix factorisation (PMF) [13] and then show how to integrate boosting procedure in PMF to learn the best feature vectors for top-N recommendations.

3.1 Probabilistic Matrix Factorisation (PMF)

Assuming there are N users and M items in the data, let matrix $R \in \Re^{N\times M}$ be a user preference matrix. PMF [13] learns two low rank matrices, user factor matrix $P \in \Re^{N\times K}$ and item factor matrix $Q \in \Re^{M\times K}$ to approximate $R \in \Re^{N\times M}$ using probabilistic inference of conditional distributions of observed rating, user priors and item priors, where K is the number of dimensions of latent factors. We use P_u to indicate the latent feature vector of user u, Q_i to indicate the latent feature vector of item i and R_{ui} to indicate the rating that user u gives to item i, respectively. The maximum of the log posterior in PMF can be formulated as

$$P, Q = argmin_{P,Q}\{\frac{1}{2}\sum_{u=1}^{N}\sum_{i=1}^{M} I_{ui}(R_{ui} - P_u Q_i^{'})^2 + \frac{\lambda_p}{2}\|P\|_F^2 + \frac{\lambda_q}{2}\|Q\|_F^2\}, \qquad (1)$$

where I_{ui} is an indicator function which equals to 1 for all observed rating, otherwise 0; λ_p and λ_q are regularisation parameters. As the user preference matrix is usually very sparse, $\|P\|_F$ and $\|Q\|_F$ are the Frobenius norms of the matrices P and Q used as regularisation to prevent the learning procedure from overfitting. We use $\lambda_p = \lambda_q = \lambda$ for computational simplicity.

3.2 BoostMF

In matrix factorisation, if one of the factor matrices, say $Q^{'}$ is fixed and only P needs to be learned, then fitting each row of the target matrix R is a linear prediction problem where $Q^{'}$ is the feature vector and each row of P is the model parameter of the linear predictor. The approximation can be formulated as a learning problem for each row $R(u,:) : R(u,:) = P(u,:) * Q^{'}$. Similarly, when P is fixed and $Q^{'}$ needs to be learned, each column of $Q^{'}$ works as the model parameter of the linear prediction model for feature vector P to fit each column of target matrix R. For each column $R(:,i)$, we have: $R(:,i) = P * Q(:,i)^{'}$. In this way, the MF can be thought as a linear regression problem where P and $Q^{'}$ are both unknown and need to be learned. Therefore, an appropriate learning algorithm to solve the linear regression problem is required.

In this work, we use boosting-based techniques to solve the linear regression problem in collaborative learning. Boosting-based techniques come with better convergence properties and stability [5]. We use boosting optimisation technique inside MF to learn low rank factor matrices directly for ranking. We aim at constructing a set of weak learners $\{F^t | t = 1, \ldots, T-1\}$ sequentially to learn user preferences and item characteristics that reside in the data. Based on latent factor selection in the weak learner construction, therefore, the algorithm will be able to stochastically focus on different aspects of user preferences and item characteristics that are modeled by the different selected latent components. By treating each rating as a training instance, a set of training weights $\{W^t | t = 1, \ldots, T-1\}$ is imposed on the ratings. An overall strong learner F is finally assembled by linearly combining weak rankers, which is expected to perform better than any individual learner. The weights of training ratings are updated to reflect the accuracy of the prediction of the weak learner. People usually follow information that appears at the top-N positions in the recommendation list. Therefore, the items that are ranked at the top should be considered more than those at the bottom of the recommendation list. To this end, we dynamically construct personalised weak rankers[1] and modify personalised weights by considering the ranking performance on training items. In next iterations, the learning procedure will give more attention on those items that have not yet been ranked in correct positions. Due to the automatic selection and optimisation of the personalised weak ranker and the dynamic updating of the personalised weights, the learned latent factors for users and items are best suited for top-N recommendation.

The BoostMF method creates weak rankers in the direction that has maximum IR performance improvement over training data. At each boosting round, the method constructs a weak ranker for each user based on IR performance over the items rated by the same user with personalised weight distribution. If user u rates m_k items and the set of items is indicated by $\mathbf{i} = i_1, i_2, \ldots, i_{m_k}$[2] then for round t, BoostMF creates a weak ranker for each user by

$$F_u^{(t)}(l) = \underset{l\in\{1,\ldots,K\}}{argmax} \left(E\left[\pi_u(W_{u\mathbf{i}}^{(t)} f_{u\mathbf{i}l}^{(t)}), R_{u\mathbf{i}} \right] \right), \tag{2}$$

where $f_{u\mathbf{i}l}^{(t)} = P_{ul}^{(t)} Q_{\mathbf{i}l}^{(t)'}$ is the ranking score according to the l-th dimension of latent factors, $W_{u\mathbf{i}}$ is the weights of user u on its item set $\mathbf{i}$, E represents the IR performance measure and K is the feature dimension of the low rank matrices, respectively. For user u, its permutation list π_u is used to order the items $\mathbf{i}$ by taking as inputs $W_{u\mathbf{i}}^{(t)}$ and $f_{u\mathbf{i}l}^{(t)}$. The design of permutation list π_u is usually correlated with the adoption of E. For simplicity, the weights $W_{u\mathbf{i}}^{(t)}$ is linearly combined with the ranking score $f_{u\mathbf{i}l}^{(t)}$ in the permutation to emphasize its confidence.

The purpose of Equation (2) is to select weak ranker for each user based on the items score $f_{u\mathbf{i}l}^{(t)}$. But in the same time we need to select the weak ranker that will be able to contribute more on items on which previous ranker did not perform well. To provide this information in weak ranker selection as well as to reflect the individual tastes and rating scale of a user, BoostMF uses weight distribution $W_{u\mathbf{i}}^{(t)}$ for each user u on every training item i from item set $\mathbf{i}$. The weight value $W_{u\mathbf{i}}^{(t)}$ is different in each round from user to user and even for the same item belongs to different user. The weight $W_{u\mathbf{i}}^{(t)}$ restricts the factor selection formula in Equation (2) not to select ranker that gives just best E measure, but to select ranker that has the ability to place the items in correct positions on which previous rankers do not perform well. BoostMF increases the values of weights on items that are not ranked well by the dynamically constructed ranking model. So in next iteration, these weight values will make more effect in next ranker selection to improve overall ranking performance.

[1] The term weak learner and weak ranker are used interchangeably throughout the paper.

[2] Bold font of i is used to denote set of items and normal to denote single item.

The weight value of an item is calculated based on the performance of the current ranker in placing the item w.r.t. other items in the ranking list. Ideally, we aim for a ranking model that makes no mistake in item placement. But the error in placing two items with rating 5 and 1 has a heavier influence on the IR performance measure than that of placing two items with rating 5 and 4. To reflect this loss, we add the pairwise preference term $(R_{ui} - R_{uj})$ into the weight function to give more penalties for misplacing the items in higher position. Specifically, if the current ranking model for user u is $F_u^{(t)}$ with selected latent factor l, and its updated ranking score on an item i is indicated by $f_{uil}^{(t+1)} = P_{ul}^{(t+1)} Q_{il}^{(t+1)'}$, the weight value of item i for that user on the l-th latent factor at $t+1$ iteration is expressed as,

$$W_{uil}^{(t+1)} = \frac{\sum_{j=1, j\neq i}^{m_k} exp\{-(f_{uil}^{(t+1)} - f_{ujl}^{(t+1)})(R_{ui} - R_{uj})\}}{\max_{i\epsilon m_k} \sum_{j=1, j\neq i}^{m_k} exp\{-(f_{uil}^{(t+1)} - f_{ujl}^{(t+1)})(R_{ui} - R_{uj})\}}. \quad (3)$$

Note that the pairwise preference term in BoostMF is different from the common pairwise preference formulation used in [9,12]. In these methods, with only implicit feedback, the pair of items consists of one observed item and one unobserved item where the observed item is assumed to have higher preference over the unobserved one. However, the formulation may be inconsistent with the real world scenario because unobserved items could be either unfavoured by the user or simply just unexposed to the user. In contrast, BoostMF uses explicit preferences to construct the personalised pairwise preference term, which is more reliable. Meanwhile, to facilitate the computation, uniform sampling is adopted in almost all of the models with the pairwise preference in order to select the set of pairs of items. However, it is shown [11] that this approach is very inefficient because most of selected items will be correctly ranked after a few of iterations and almost all the gradient magnitude from the selected pairs become less informative. In this regards, BoostMF provides an efficient and informative selection and updating mechanism by constantly focusing on the disordered items for every user across the whole procedure of learning.

At the initialisation, the value of user weight $W_{ui}^{(t)}$ on every item is identical. At the current round t, BoostMF increases the values of weights on items that are not ranked well by the dynamically constructed ranking model. Hence in round $t+1$ these weights will make more contribution to construct the next ranking model that will attempt to rectify the incorrect ranking of these items. The value of $W_{ui}^{(t)}$ yields a clear indication how much the item i is misplaced in the rank list of user u. So in next iteration, this weight will make more effect in next ranker selection to improve the performance.

To model the fact that various users will judge their preferences over different items based on different criteria, BoostMF also selects the direction that has maximum capacity to generate a good ranking list on the training items for each user at every round and performs maximum adjustment in that direction. All other factors for that user are remaining unchanged on the round. Let l denote the dimension of the selected latent factors for the current weak ranker. If the objective function in Equation (1) is denoted by L, then for the ranking

Algorithm 1. Boosted Matrix Factorisation

Input: Rating matrix R, no. of iterations T, performance measure E, no. of users N and no. of items M, no. of training items per user m_k, feature dimension K and learning rate η
Output: Low rank factor matrices P and Q
Initialisation: Initialise $P^{(1)}$ and $Q^{(1)}$ randomly, and initialise $W^{(1)}_{u\mathbf{i}} = \frac{1}{m_k}$ for each user u on available training items.

```
for t=1:T-1 do
  for u=1:N do
    Select ranking model $F^{(t)}_u(l)$, $l \in \{1, \dots, K\}$ using Equation (2) for user $u$ on its rated item set $\mathbf{i}$ with weighted distribution of $W^{(t)}_{u\mathbf{i}}$.
    Compute $\frac{\delta L}{\delta P^{(t)}_{ul}}$ and $\frac{\delta L}{\delta Q^{(t)}_{\mathbf{i}l}}$ using Equation (4) and (5).
    Update $P_{ul}$ and $Q_{\mathbf{i}l}$ by
    $P^{(t+1)}_{ul} = P^{(t)}_{ul} - \eta \frac{\delta L}{\delta P^{(t)}_{ul}}$, $Q^{(t+1)}_{\mathbf{i}l} = Q^{(t)}_{\mathbf{i}l} - \eta \frac{\delta L}{\delta Q^{(t)}_{\mathbf{i}l}}$
    Update $W^{(t+1)}_{u\mathbf{i}}$ using Equation (3).
  end for
end for
```

Output: $P^{(T)} Q^{(T)'}$

model $F^{(t)}_u$, BoostMF updates user and item latent factor by

$$\frac{\delta L}{\delta P^{(t)}_{ul}} = \sum_{i=1}^{m_k} I_{ui}(P^{(t)}_{ul} Q^{(t)'}_{il} - R_{ui}) Q^{(t)}_{il} + \lambda P^{(t)}_{ul} \tag{4}$$

$$\frac{\delta L}{\delta Q^{(t)}_{\mathbf{i}l}} = I_{u\mathbf{i}}(P^{(t)}_{ul} Q^{(t)'}_{\mathbf{i}l} - R_{u\mathbf{i}}) P^{(t)}_{ul} + \lambda Q^{(t)}_{\mathbf{i}l}.[3] \tag{5}$$

Compared with the updating stages of latent factors in conventional MF methods, the difference terms in Equation (4) and Equation (5) in BoostMF have also shifted the focus to the contribution of individual latent factor. Instead of combining the weak learner estimation to form final strong learner, BoostMF takes all latent dimensions of $P^{(T)} Q^{(T)'}$ as strong learner after the completion of round $T-1$, as the weak ranking models are updated during learning. Finally, personalised ranking list is generated by sorting the ratings which are predicted by using all latent dimensions of $P^{(T)}$ and $Q^{(T)'}$. At each boosting round $t = 1, \dots, T-1$, BoostMF creates a weak ranker $F^{(t)}_u$ for each user, updates the ranker, modifies weights based on the ranking performance and finally outputs a personalised ensemble model as $F \approx P^{(T)} Q^{(T)'}$. An overview of the algorithm is presented in Algorithm 1.

The complexity of weak ranker selection is in the order of $\mathcal{O}(NK)$, where N is the number of users and K is the feature dimension size. The complexity of the gradient computation in Equation (4) and (5) is in order of $\mathcal{O}(R)$, where R is the number of observed ratings in the given user-item matrix. The computation complexity of the weight updating formula in Equation (3) is $\mathcal{O}(Nm_k^2)$, where m_k denotes the number of training items per user. In collaborative filtering, $R \gg N, M$ and even dominates

[3] The summation sign is not used on the right hand side of (5), because we formulate the algorithm user-wise and the item set i is rated by one user as shown in Algorithm 1.

the term Nm_k^2. When Nm_k^2 dominates R, BoostMF has complexity in the order of $\mathcal{O}(Nm_k^2)$, otherwise it has linear time complexity in the order of $\mathcal{O}(R)$.

3.3 Theoretical Analysis

In this section, we show theoretical insights by developing an upper error bound of BoostMF following MPBoost [19] and RankBoost [4]. Allowing both rating magnitude in the ranking loss and dynamic changes in the feature vectors for the weak learner model at each boosting round in BoostMF leads to the following theorem:

Theorem 1. *The misplacement loss of the personalised ranking model in BoostMF is bounded by* $\sum_{i=1}^{m_k}\sum_{j=1,j\neq i|R_i>R_j}^{m_k} [\![F_i \leq F_j]\!] + \sum_{i=1}^{m_k}\sum_{j=1,j\neq i|R_i<R_j}^{m_k} [\![F_i \geq F_j]\!] \leq Z_T$, *where* $Z_T = \sum_{i=1}^{m_k}\sum_{j=1,j\neq i}^{m_k} exp\{-[(F_i - F_j)(R_i - R_j)]\}$ *and* $[\![x]\!]$ *is defined to be 1 if predicate* x *is true and 0 otherwise.*

Proof. The personalised ranking model in BoostMF produces two types of misplacement. The first one is when $F_i \geq F_j$ but $R_i < R_j$ and the second one is when $F_i \leq F_j$ but $R_i > R_j$. Note that $[\![x \geq 0]\!] \leq exp\{\alpha x\}$ and $[\![x \leq 0]\!] \leq exp\{-\alpha x\}$ hold for all $\alpha > 0$ and all real x. We can write the total loss as

$$\begin{aligned}
&\sum_{i=1}^{m_k}\sum_{j=1,j\neq i|R_i>R_j}^{m_k} [\![F_i \leq F_j]\!] + \sum_{i=1}^{m_k}\sum_{j=1,j\neq i|R_i<R_j}^{m_k} [\![F_i \geq F_j]\!] \\
&\leq \sum_{i=1}^{m_k}\sum_{j=1,j\neq i|R_i>R_j}^{m_k} exp\{-[(F_i - F_j)(R_i - R_j)]\} \\
&+ \sum_{i=1}^{m_k}\sum_{j=1,j\neq i|R_i<R_j}^{m_k} exp\{[-(F_i - F_j)(R_i - R_j)]\} \\
&= \sum_{i=1}^{m_k}\sum_{j=1,j\neq i}^{m_k} exp\{-[(F_i - F_j)(R_i - R_j)]\} = Z_T
\end{aligned}$$ ■

This bound is guaranteed to produce a combined low ranking loss if we choose the weak ranker that minimises $\sum_{i=1}^{m_k}\sum_{j=1,j\neq i}^{m_k} exp\{-[(F_i - F_j)(R_i - R_j)]\}$ on each round t [4]. Minimising the misplacement loss is equivalent to maximising the IR measure [19]. In BoostMF, the weak ranker is set to select the ranking model that maximises the IR measure which is equivalent to minimising the misplacement loss and the weight update formula is set to give more penalties to the ranking model that makes misplacement in higher position. Finally, in terms of misplacement loss, the total error of BoostMF is bounded by Z_T.

4 Experiments

4.1 Datasets and Evaluation Metric

We test the performance of BoostMF on three publicly available datasets for the task of personalised top-N recommendation: MovieLens 100K[4] dataset, MovieLens 1M dataset[4] and Netflix[5] dataset. MovieLens 100K dataset consists of 100,000 ratings from 943 users on 1682 movies. MovieLens 1M dataset consists of 1,000,000 ratings from 6040 users and 3900 movies. Ratings are integers and scaled on 1-5. and each user has rated at least 20 movies on both datasets. For Netflix dataset, we use a sampled version, which is extracted from 4% of the

[4] http://www.grouplens.org/node/73
[5] B. James and L. Stan, The Netflix prize, (2007).

Netflix dataset with 20% users and 20% movies are randomly selected from the whole pool. The Netflix dataset contains 3,843,340 ratings on scaled 1-5 from 95526 users on 3561 movies.

As our goal is to generate efficient recommendation list that would contain higher rating items in top-N position, we prefer a metric capable of awarding models that correctly rank items in higher positions and penalising models that make more errors in higher positions than in lower positions. Following the standard evaluation metric used in [2,15,17], we use normalised discounted cumulative gain (NDCG) as IR performance measure for testing and evaluation of our algorithm.

4.2 Experimental Setup

We adopt the same experimental protocol from [2,17]. We use 3 different settings of training data based on the number of randomly selected items for each user, namely S_N=10, S_N=20 and S_N=50. The remaining items are used for testing. Users with less than 20, 30 or 60 rated items are removed respectively in each setting to ensure the feasibility to compute NDCG@10. Following the common practice in RS [2,17], items that are not rated by at least 5 users in the dataset are also removed. We also eliminate items from test dataset those are not appeared in training dataset. These settings cause a slightly decrement of user-movie combination than the original dataset. We report the number of users and items available in each setting for all datasets in Table 1. For each setting, we generate 10 versions of the dataset, by randomly sampling items. We report the mean and standard deviation of NDCG@5 and NDCG@10 on those 10 sets over all users. We compare BoostMF with a sets of the state-of-the-art algorithms including PMF [13], and OrdRec [8] which are the state-of-the-art rating-oriented CF methods; ListRank-MF [15], CofiRank [17], and BPR-MF [12], which are the state-of-the-art ranking-oriented CF methods; and URM [16], which combines both rating-oriented and ranking-oriented methods. From the experimental results in [17], CofiRank method that optimises root mean square loss (denote as CofiRank$^{\text{Reg}}$) performs better than CofiRank that optimises NDCG directly (denote as CofiRank$^{\text{NDCG}}$). Therefore, we compare BoostMF with both CofiRank$^{\text{Reg}}$ and CofiRank$^{\text{NDCG}}$.

Table 1. No. of users and items for experimental settings S_N=10, 20 and 50 on the datasets.

Dataset	No. of users for S_N=10/20/50	No. of items for S_N=10/20/50
MovieLens 100K	941/743/496	1349/1336/1312
MovieLens 1M	6035/5286/3937	3415/3411/3400
Netflix	45508/35749/20067	3558/3556/3546

4.3 Results

Before comparing the performance of our algorithm with other state-of-the-art approaches, we at first examine whether the weak ranker selection and weight update formula in BoostMF improve the algorithms performance or not. To this end, we create two versions of BoostMF algorithm named (1) RandomBoostMF that selects weak ranker randomly for each user, (2) ModifiedBoostMF that selects ranking model by NDCG but updates item weights without considering the effect of the pairwise preference term $(R_{ui} - R_{uj})$. The comparison of BoostMF with RandomBoostMF indicates whether factor selection by NDCG in BoostMF makes any benefits over random factor selection, and the comparison with ModifiedBoostMF indicates the advantages of using the modified weight update mechanism in Equation (3). We also want to see how these algorithms perform with respect to various feature dimensions. To apply these algorithms, MovieLens 100K dataset with user/item settings S_N=50 is used. We record NDCG@10 for each data fold for factor dimension 5, 10, 15, 25 and 50 and the mean of NDCG over 10 folds for each feature dimension is presented in Fig. 1.

From the results in Fig. 1, we can see that BoostMF performs much better than RandomBoostMF, which verifies that factor selection by NDCG in BoostMF helps improve the performance of top-N recommendation as random selection is not able to generate suitable feature vectors to boost the learning procedure of MF. BoostMF also outperforms ModifiedBoostMF, which shows the success of the developed pairwise preference scheme in the procedure of dynamic weight updating. Most importantly, the performance of BoostMF is stable under different settings of the feature dimension size. It performs the best for feature dimension size of 15 which the NDCG score is 0.7139. The NDCG score slightly decreases for feature dimension size of 50 which is 0.7020 but still much better in comparison to the performance of RandomBoostMF (0.6765) and ModifiedBoostMF (0.6908).

Now we compare our algorithm with PMF, ListRank-MF, URM, OrdRec, BPR-MF and CofiRank. We tune parameters separately on a validation set for all algorithms by cross validation to achieve their best performance on the used datasets. NDCG performance on validation set is used to choose the hyperparameters with the best performance. We implement CofiRank using publicly available software.[6] OrdRec[8] and BPR-MF[12] are implemented by publicly available software Lenskit[7] and Mymedialite[8], respectively. BoostMF uses η=0.01, λ=0.02 and K=5 for MovieLens datasets and η=0.00005, λ=0.000009 and K=10 for Netflix dataset. As each method has different settings of hyperparameters under different settings of experiments, due to the space limitation, we do not state the hyperparameters of other algorithms. We also perform paired t test [6] with significant level of 5%, and all the improvement are statistically significant. The mean and standard deviation over 10 data folds for different approaches with respect to different experimental settings are reported in Table 2-4.

[6] http://www.cofirank.org/downloads.

[7] http://lenskit.org/download/

[8] http://mymedialite.net/download/index.html

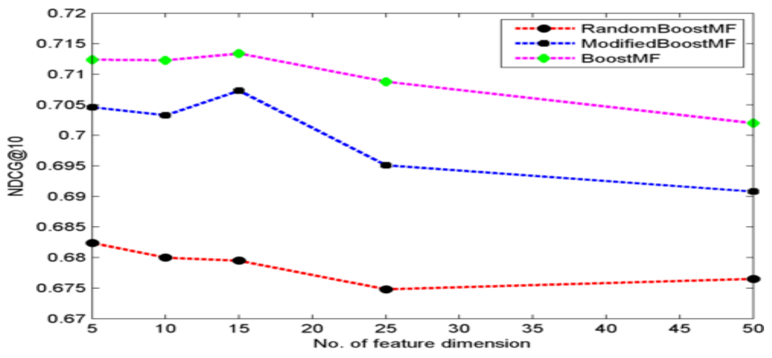

Fig. 1. Performance comparison of RandomBoostMF, ModifiedBoostMF and BoostMF.

According to Table 2, BoostMF significantly outperforms all compared state-of-the-art algorithms in most of the cases. BoostMF achieves 10∼12% improvement over CofiRank$^{\mathrm{NDCG}}$ for settings S_N=50 and gains 6∼8% improvement for settings S_N=10 and S_N=20 on both NDCG@5 and NDCG@10 metrics for MovieLens 100K dataset. It also gains 1.6∼4% improvement over PMF, 1∼4.7% improvement over URM and 3∼5% improvement over CofiRank$^{\mathrm{Reg}}$ on both evaluation measures for all experimental settings. BoostMF also shows 0.7∼2.3% improvement over ListRank-MF. Although for settings S_N=10, BoostMF performs slightly worse than ListRank-MF on NDCG@5, it performs better than ListRank-MF for all other settings on both metrics. BoostMF outperforms BPR-MF and OrdRec for all experimental settings on both NDCG@5 and NDCG@10 metrics. It achieves 8∼11% improvement over BPR-MF and 10∼20% improvement over OrdRec.

Results on MovieLens 1M dataset are shown in Table 3. BoostMF achieves significant improvement over CofiRank, BPR-MF and OrdRec for all experimental settings on all the evaluations. It achieves 10-19% improvement over OrdRec, 5-8% improvement over BPR-MF, and 4-9% improvement over CofiRank$^{\mathrm{NDCG}}$ and CofiRank$^{\mathrm{Reg}}$ respectively on both evaluations for all experimental settings. In comparison with URM, BoostMF achieves 2∼2.4% improvement on NDCG@5 metric and 1.1∼2.14% improvement on NDCG@10 metric over all experimental settings. BoostMF also outperforms ListRank-MF by 9∼10% for settings S_N=10, 6.5∼6.8% for setting S_N=20 and also achieves more than 1% improvement for settings S_N=50 on both evaluations. It gains 5∼6% improvement for settings S_N=10 and 3∼4% improvement for settings S_N=20 and S_N=50 on both metrics when comparison is made with PMF.

From the results in Table 4, it is clear that BoostMF outperforms all other state-of-art approaches on Netflix dataset. It outperforms CofiRank$^{\mathrm{NDCG}}$ by 10∼12% over all experimental settings on both NDCG computations. It achieves 7∼10% improvement over CofiRank$^{\mathrm{Reg}}$ for experimental settings S_N=10, S_N=20 and 4∼5% improvement for experimental settings S_N=50 on both metrics. Compared to OrdRec and BPR-MF, BoostMF results 8∼16% improvement for all settings on both metrics. It also gains 9∼14% performance improvement for

Table 2. The NDCG@5 and NDCG@10 accuracy and standard deviation over 10 data folds for PMF, BPR-MF, ListRank-MF, URM, OrdRec, CofiRank and BoostMF on MovieLens 100K dataset. The best performance is in bold.

	S_N=10		S_N=20		S_N=50	
	NDCG@5	NDCG@10	NDCG@5	NDCG@10	NDCG@5	NDCG@10
PMF	0.6330±.009	0.6606±.005	0.6762±.007	0.6864±.007	0.6765±.005	0.6819±.007
BPR-MF	0.5558±.002	0.5942±.003	0.5872±.002	0.6098±.004	0.6292±.001	0.6309±.002
CofiRank$^{\mathrm{NDCG}}$	0.5927±.006	0.6314±.006	0.6098±.005	0.6331±.003	0.5897±.006	0.6096±.005
CofiRank$^{\mathrm{Reg}}$	0.6381±.008	0.6629±.004	0.6398±.003	0.6540±.004	0.6580±.004	0.6708±.002
OrdRec	0.5197±.001	0.5687±.001	0.4852±.003	0.5290±.002	0.58±.002	0.6081±.004
ListRank-MF	**0.6725±.005**	0.6844±.005	0.6834±.004	0.6947±.003	0.6887±.003	0.6982±.004
URM	0.6421±.005	0.6561±.006	0.6778±.004	0.6851±.007	0.6919±.005	0.7034±.004
BoostMF	0.6722±.008	**0.7034±.007**	**0.6921±.005**	**0.7019±.004**	**0.7117±.004**	**0.7135±.004**

Table 3. The NDCG@5 and NDCG@10 accuracy and standard deviation over 10 data folds for PMF, BPR-MF, ListRank-MF, URM, OrdRec, CofiRank and BoostMF on MovieLens 1M dataset. The best performance is in bold.

	S_N=10		S_N=20		S_N=50	
	NDCG@5	NDCG@10	NDCG@5	NDCG@10	NDCG@5	NDCG@10
PMF	0.6814±.007	0.6842±.005	0.7030±.002	0.7043±.003	0.7224±.002	0.7189±.003
BPR-MF	0.6734±.007	0.6873±.006	0.6747±.005	0.6790±.006	0.6711±.007	0.6769±.006
CofiRank$^{\mathrm{NDCG}}$	0.6485±.002	0.6685±.002	0.6587±.005	0.6763±.001	0.6679±.007	0.6812±.007
CofiRank$^{\mathrm{Reg}}$	0.6698±.005	0.6838±.006	0.6728±.005	0.7005±.005	0.6844±.007	0.7049±.006
OrdRec	0.5095±.002	0.5431±.003	0.4948±.002	0.5312±.002	0.6288±.002	0.6485±.001
ListRank-MF	0.6424±.005	0.6423±.004	0.6792±.007	0.6827±.006	0.7406±.004	0.7344±.004
URM	0.7205±.004	0.7222±.002	0.7236±.001	0.7365±.001	0.7328±.003	0.7301±.002
BoostMF	**0.7433±.007**	**0.7389±.007**	**0.7475±.005**	**0.7480±.004**	**0.7528±.004**	**0.7515±.004**

settings S_N=10, 7∼8% for settings S_N=20 and more than 4% for settings S_N=50 over PMF on both NDCG evaluations. Over ListRank-MF, BoostMF gains 1.6∼3.8% improvement on NDCG@5 metric for settings S_N=20 and S_N=50, and it gains 7∼10% improvement on NDCG@10 for experimental settings S_N=10. It also outperforms URM by 2∼4.8% on NDCG@5 computation and 1.4∼2.8% on NDCG@10 computation for all experimental settings.

To gain a deep understanding of the success of BoostMF, the reasons for the experimental results will be explored as follows. PMF is the rating-oriented collaborative filtering algorithm that minimises sum of squared errors at each step of learning process. Hence, the learning procedure spends its efforts on a criterion that is not directly related to the task of top-N recommendation. OrdRec, which is a regression based rating-oriented CF method, assumes users' feedback as ordinal rather than number. Although it considers users' personalised rating scales, the ranking measure is not directly applied to the learning model. ListRank-MF is the ranking-oriented CF algorithm that aims to present better ranking list, however, unlike BoostMF, the IR evaluation measure of ListRank-MF is not directly related to the learning process of the model. BPR-MF, which is a ranking

Table 4. The NDCG@5 and NDCG@10 accuracy and standard deviation over 10 data folds for PMF, BPR-MF, ListRank-MF, URM, OrdRec, CofiRank and BoostMF on Netflix dataset. The best performance is in bold.

	S_N=10		S_N=20		S_N=50	
	NDCG@5	NDCG@10	NDCG@5	NDCG@10	NDCG@5	NDCG@10
PMF	0.6239±.007	0.5913±.006	0.6481±.004	0.6689±.004	0.6876±.007	0.6980±.005
BPR-MF	0.5724±.003	0.6005±.005	0.5725±.004	0.5982±.005	0.5783±.003	0.6095±.003
CofiRank$^{\mathrm{NDCG}}$	0.6203±.001	0.6272±.005	0.6134±.002	0.6139±.003	0.6249±.005	0.6316±.002
CofiRank$^{\mathrm{Reg}}$	0.6146±.005	0.6128±.004	0.6252±.003	0.6599±.005	0.6790±.006	0.6931±.005
OrdRec	0.5597±.002	0.6076±.003	0.5908±.005	0.6315±.006	0.6264±.002	0.6556±.003
ListRank-MF	0.6453±.002	0.6359±.001	0.7011±.004	0.7017±.007	0.7156±.007	0.7118±.002
URM	0.6808±.006	0.7217±.002	0.7118±.002	0.7188±.005	0.6831±.002	0.7099±.004
BoostMF	**0.7216±.006**	**0.7364±.002**	**0.7352±.004**	**0.7398±.006**	**0.7317±.003**	**0.7383±.002**

based model, solves personalised ranking problem by optimising area under the curve (AUC). Unlike the optimisation criterion used in BoostMF, AUC imposes equal error on misplacing items irrespective of their positions in the generated recommendation list; thus BPR-MF does not perform well on list based top-N recommendation. URM employs both rating and ranking information together but still the IR evaluation measure is not directly applied in the learning model. Meanwhile, the relative contribution of rating information and ranking information depends on the particular dataset. CofiRank is also a ranking-oriented CF method, but from our experimental results, CofiRank$^{\mathrm{NDCG}}$ that uses NDCG information directly into learning phases performs worse than CofiRank$^{\mathrm{Reg}}$ that optimises for regression with root mean square loss. This finding is consistent with experimental results from [2,15,17]. On the other hand, BoostMF is proposed to improve the underlying factor learning in matrix factorisation using boosting with feature selection and to optimise IR measure directly aiming at resolving the mismatch between training objective function and evaluation metric. Without imposing any overhead of IR measure conversion, BoostMF creates ranking model and updates according to the correctness of the ranking list. Thus, BoostMF presents a better ranked recommendation list than the state-of-the-art recommendation approaches by focusing on the factors that are best suited to represent ranking task.

In addition to the comparison, we carry the experiment to see in what extent PMF and BoostMF handle overfitting, which is an important issue when the data is extremely sparse which is common in recommender systems. Note that this is also important for the boosting algorithm whose capability of generalisation is usually considered under non-sparse dataset [4,19]. Specifically, we want to evaluate the IR performance of both algorithms, i.e. PMF and BoostMF, on the test set while the IR performance keeps increasing on training set as the number of training round increases. For experimental settings of S_N=50 on MovieLens 100K dataset, we record the performance of the models on both train set and test set for every iteration of PMF and BoostMF. Fig. 2 and 3 show the average NDCG@10 on train set and test set over 10 folds. Learning rate and

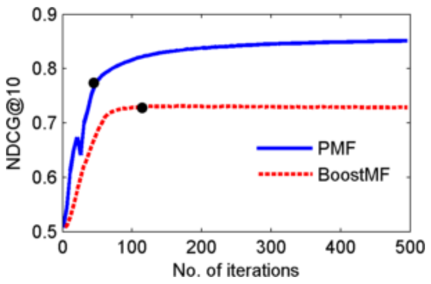

Fig. 2. Average NGCG@10 of PMF and BoostMF over 10 folds on train set

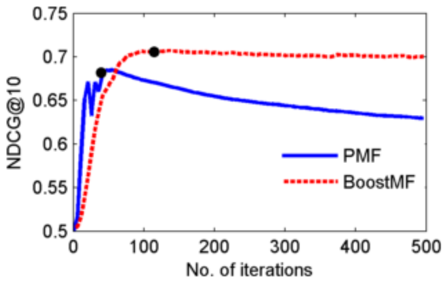

Fig. 3. Average NGCG@10 of PMF and BoostMF over 10 folds on test set

regularisation parameter of PMF and BoostMF are set separately according to their best performance on validation set. The stopping conditions for both algorithms are also set from cross validation and marked as black circle in Fig. 2 and 3.

As shown in the Fig. 2 and 3, PMF suffers from serious overfitting problem whereas BoostMF is very robust to overfitting. This overfitting behaviour of PMF shows its inappropriateness to the ranking problem. As the iteration continues, it ignores the information from IR measure and thus deviates away from improving NDCG. The test performance in PMF is decreasing while the training performance is steadily improving. On the contrary, BoostMF constructs weak learner in the direction that gives maximum ranking accuracy on training data, performs maximum update in that ranking direction and reweights items according to the correctness of the ranking list. All of those three features are the keys for stable ranking as the number of training round increases and thus it avoids overfitting issues.

5 Conclusion

In this paper, we present a novel method, BoostMF, to the problem of matrix factorisation by learning the best feature vectors for ranking and apply it to the task of personalised top-N recommendation. In addition to using latent factors to represent various user preferences and item characteristics, the BoostMF method uses boosting procedure to select best factors to optimise for the ranking task and performs updating only on that factor. In contrast to other ranking-oriented CF methods, the BoostMF method optimises the ranking measure directly by learning low rank factor matrices rather than using the structured estimation of ranking loss or computing continuous approximations of IR measure. To demonstrate the efficiency of BoostMF, we evaluate it against a set of state-of-the-art approaches on three real-world publicly available datasets with different user-item distributions. The experimental results verify that the BoostMF method achieves significant improvement over these baseline methods for the task of top-N recommendation.

Our method will cope with cold start user problems in the future that users have very few ratings (one or two). We would like to apply our boosting-based collaborative filtering model with other IR evaluation metrics, such as minimum average precision (MAP) and minimum reciprocal ranking (MRR) for recommendation. We also want to apply this algorithm to other problems where MF approach is frequently applied. As MF consists of the fundamentals of many existing methods in top-N recommendation, it is reasonable to expect the proposed method also to be valuable for existing top-N recommendation methods that are based on PMF such as these methods shown in Section 2.

References

1. Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. IEEE TKDE **17**(6) (2005)
2. Balakrishnan, S., Chopra, S.: Collaborative ranking. In: WSDM (2012)
3. Cai, X., Bain, M., Krzywicki, A., Wobcke, W., Kim, Y., Compton, P., Mahidadia, A.: Learning collaborative filtering and its application on people-to-people recommendation in social networks. In: ICDM (2010)
4. Freund, Y., Iyer, R., Schapire, R., Singer, Y.: An efficient boosting algorithm for combining preferences. J. Mac. Learn. Res. **4** (2003)
5. Friedman, J., Hastie, T., Tibshirani, R.: Additive Logistic Regression: a Statistical View of Boosting. The Annals of Statistics **38**(2) (2000)
6. Goulden, C.: Methods of Statistical Analysis. Wiley (1956)
7. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. J Comput. **42** (2009)
8. Koren, Y., Sill, J.: Ordrec: an ordinal model for predicting personalized item rating distributions. In: RecSys (2011)
9. Krohn-Grimberghe, A., Drumond, L., Freudenthaler, C., Schmidt-Thieme, L.: Multi-relational matrix factorization using bayesian personalized ranking for social network data. In: WSDM (2012)
10. Liu, T.: Learning to rank for information retrieval. Found. and Trends in Inf. Retr. **3** (2009)
11. Rendle, S., Freudenthaler, C.: Improving pairwise learning for item recommendation from implicit feedback. In: WSDM (2014)
12. Rendle, S., Freudenthaler, C., Gantner, Z., Thieme, L.: BPR: bayesian personalized ranking from implicit feedback. In: UAI (2009)
13. Salakhutdinov, R., Mnih, A.: Probabilistic matrix factorization. In: NIPS (2008)
14. Shi, Y., Karatzoglou, A., Baltrunas, L., Larson, M., Oliver, N., Hanjalic, A.: CLiMF: Learning to maximize reciprocal rank with collaborative less-is-more filtering. In: RecSys (2012)
15. Shi, Y., Larson, M., Hanjalic, A.: List-wise learning to rank with matrix factorization for collaborative filtering. In: RecSys (2010)
16. Shi, Y., Larson, M., Hanjalic, A.: Unifying rating-oriented and ranking-oriented collaborative filtering for improved recommendation. J. of Inf., Sci. (2013)
17. Weimer, M., Karatzoglou, A., Le, Q.V., Smola, A.: CofiRank-maximum margin matrix factorization for collaborative ranking. In: NIPS (2007)
18. Xu, J., Li, H.: AdaRank: a boosting algorithm for information retrieval. In: SIGIR (2007)
19. Zhu, C., Chen, W., Zhu, Z., Gang, W., Wang, D., Chen, Z.: A general magnitude-preserving boosting algorithm for search ranking. In: CIKM (2009)