Collaborative Ranking with a Push at the Top

Konstantina Christakopoulou
Dept. of Computer Science & Engineering
University of Minnesota, Twin Cities
christa@cs.umn.edu

Arindam Banerjee
Dept. of Computer Science & Engineering
University of Minnesota, Twin Cities
banerjee@cs.umn.edu

ABSTRACT

The goal of collaborative filtering is to get accurate recommendations at the top of the list for a set of users. From such a perspective, collaborative ranking based formulations with suitable ranking loss functions are natural. While recent literature has explored the idea based on objective functions such as NDCG or Average Precision, such objectives are difficult to optimize directly. In this paper, building on recent advances from the learning to rank literature, we introduce a novel family of collaborative ranking algorithms which focus on accuracy at the top of the list for each user while learning the ranking functions collaboratively. We consider three specific formulations, based on collaborative p-norm push, infinite push, and reverse-height push, and propose efficient optimization methods for learning these models. Experimental results illustrate the value of collaborative ranking, and show that the proposed methods are competitive, usually better than existing popular approaches to personalized recommendation.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; I.2.6 [**Computing Methodologies**]: Artificial Intelligence—*Learning*

Keywords

Recommender systems; collaborative ranking; infinite push

1. INTRODUCTION

Personalized recommendation has become popular in a wide range of applications, such as electronic commerce, web search, news discovery, and more [2, 6, 13, 23]. Typically, the recommendation system selects a small set from the underlying pool of items (e.g., news, movies, books) to recommend to the user. The user can either like (e.g., read, click, buy, etc.) or dislike the items in the list, or take no action. Ideally, we want each user to like the items presented towards the top of the ranked list, since these are the ones she will actually get to see.

Collaborative filtering is a popular approach for recommendations systems. Typically such techniques assume that the user-item rating matrix R is low-rank, and they collaboratively learn the latent factors U and V for the users and the items correspondingly [19, 25]. Probabilistic matrix factorization type of approaches, widespread especially after the Netflix challenge [6], pose the problem of personalized recommendation as a rating prediction or matrix completion problem, trying to predict the ratings of unrated items [1, 19, 25, 31, 33]. Such approaches consider the square loss or variants as a measure of prediction accuracy. In reality, for a 5-point rating scale, predicting a true 5 as a 3 is often more costly than predicting a 3 as a 1, although their square loss is the same. More generally, what matters in a recommendation system is the ranking of items, especially the ranking performance at the top of the list, i.e., the items the user will actually see.

In this paper, we consider the recommendation problem as a collaborative ranking (CR) problem, such that the ranking loss focuses on accuracy at the top of the list for each user. The consideration is inspired by recent results [5, 38, 39] which support the hypothesis that posing recommendation problems as a ranking problem, as opposed to matrix completion or ratings prediction, leads to better performance. While existing CR models do optimize certain classical ranking metrics [34, 35, 36, 39], most of these metrics are non convex, so one works with a relaxation of the original problem, e.g., CofiRank [39] optimizes a convex upper bound of a loss based on Normalized Discounted Cumulative Gain (NDCG) [17], CLiMF [34] optimizes a smoothed version of the Reciprocal Rank [4] via a lower bound, etc. Moreover, certain CR models [22] consider the entire list, rather than focusing on accuracy at the top of the list.

We build on two good ideas respectively from the collaborative filtering and the learning to rank literature. First, the low rank representation with latent factor for users and items is indeed a simple and powerful idea. Second, the convex loss functions developed in the recent literature on learning to rank, such as p-norm push, infinite push, and reverse-height push [3, 28, 30], perform well to get accurate rankings at the top of the list. In our work, we use low rank representations for CR while working with ranking losses for individual users which focus on accuracy at the top of the list for each user. The result is a flexible new family of approaches for CR based on collaborative p-norm push, infinite

WWW 2015, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3469-3/15/05.
http://dx.doi.org/10.1145/2736277.2741678.

push, and reverse-height push. We propose efficient alternating minimization methods for iteratively updating the latent factors, including a new approach for the non-smooth problem in infinite push based on gradient mapping for minimax problems [26]. Empirical evaluation carefully considers three key questions: (i) Between square loss and ranking loss with focus at the top of the list, which does better with recommendations at the top of the list? (ii) Between individual ranking and collaborative ranking, which does better for recommendations across all users, and (iii) Are the new collaborative ranking methods competitive with related existing ideas? Through extensive experiments, we illustrate that ranking loss indeed performs better than squared loss, collaborative ranking outperforms individual ranking, and the proposed set of algorithms are competitive, usually better than existing baselines.

The rest of the paper is organized as follows. We briefly review related work in Section 2. In Section 3, we combine the ideas of push and collaborative ranking, and propose three algorithms for collaborative ranking with a push at the top of the list. We empirically evaluate the proposed methods in Section 4, and conclude in Section 5.

2. RELATED WORK

In Information Retrieval (IR), the Learning To Rank (LTR) problem considers a set of training queries, where for each query a set of retrieved documents along with their relevance scores are given [8, 9, 10, 11, 24, 32]. The goal is to learn a ranking function which maps the features of the documents to a relevance score, based on which the documents will be ranked by degree of predicted relevance. A vast literature for LTR exists, and the categories of the methods proposed to approach the problem are *pointwise* [12], *pairwise* [3, 7, 8, 9, 30, 32] and *listwise* [10, 40]. The *pointwise* techniques require fitting to the actual relevance values, which is analogous to optimizing for the AUC metric. Thus, pointwise techniques try to predict the exact relevance scores of unseen documents instead of just capturing the correct ordering between them. The *pairwise* approach avoids these drawbacks by predicting the preference order between data. The two LTR approaches we employ, i.e., P-Norm Push [30] and Infinite Push [3] both lie in the category of pairwise LTR approaches. Their main disadvantage is the computational complexity, which is however not prohibitive when being applied to recommender systems, because of the sparsity of ratings for the users. *Listwise* approaches learn a ranking model for the entire set of documents.

Inspired by the analogy between query-document relations in IR and user-item relations in recommender systems, many CR approaches have been proposed over the past few years for personalized recommendations [5, 20, 22, 29, 34, 35, 36, 38, 39]. The majority of them optimize one of the (top-N) ranking evaluation metrics by exploiting advances in structured estimation [37] that minimize convex upper bounds of the loss functions based on these metrics [21]. CofiRank is one of the state-of-the-art listwise CR approaches which optimizes a convex relaxation of the NDCG measure for explicit feedback data (i.e., ratings) [39]. CLIMF and xCLIMF respectively optimize a smooth lower bound for the mean reciprocal rank on implicit (based on user behavior) feedback data [34], and expected reciprocal rank for data with multiple levels of relevance [36]. GAPfm [35] has been proposed to optimize Graded Average Precision, in order to address the concerns with Average Precision in multiple-relevance graded data. Recent work in [38], in the same spirit as [5], argued the need to bridge the ideas in the LTR community with Collaborative Filtering (CF). For that, they used a novel approach of generating features for the LTR task using CF techniques, i.e., neighborhood-based and model-based techniques, so that the problem can be transformed to a classic pairwise LTR problem. Recently, [22] introduced a method for Local Collaborative Ranking (LCR) where ideas of local low-rank matrix approximation were applied to the pairwise ranking loss minimization framework. A special case of this framework is Global Collaborative Ranking (GCR) where global low-rank structure is assumed, as is usually done in the ranking counterpart of Probabilistic Matrix Factorization (PMF)-type methods [1, 25, 31, 33], although equal importance is given to the whole ranked list.

3. PUSH COLLABORATIVE RANKING

Consider a collaborative ranking setting with m users $\{\rho_1, \ldots, \rho_m\}$ and n items $\{x_1, \ldots, x_n\}$, where the items can be movies, books, news articles, etc. Each user ρ_i is assumed to have rated a total of $n_i \leq n$ items, of which n_i^+ items are rated as 'relevant' and n_i^- items are rated as 'not relevant.' Let L_i^+ denote the set of relevant items and L_i^- denote the set of non-relevant items for user ρ_i, so that $|L_i^+| = n_i^+$ and $|L_i^-| = n_i^-$. For convenience, for a given user ρ_i, we will denote the relevant items as x_k^+ (not $x_k^{i,+}$ to avoid clutter) and the non-relevant items as x_j^-.

The goal is to construct ranking/scoring functions $f_i(x)$ over the items for each user ρ_i such that relevant items for ρ_i get a higher rank/score. From a ranking perspective, $f_i(x)$ induces a ranking over all items, and the goal is to make sure relevant items are ranked higher than non-relevant items. To accomplish the goal, our algorithms focus on two aspects: the ranking functions $f_i(x)$ for users will be

(i) trained with focus on accuracy at the top of the list for each user, and
(ii) constructed collaboratively across all the users, rather than separately for each user.

The working hypothesis that emphasis on these two aspects for collaborative ranking lead to better performance will be carefully evaluated in Section 4, where we consider alternative approaches which give equal importance to the entire list, or train the ranking functions individually on each user. We start with a discussion on how these two aspects are realized in our algorithms.

3.1 Push and Collaborative Ranking

Ranking with a Push: The idea of ranking with emphasis on the top of the list has been explored in the learning to rank literature [3, 7, 28, 30]. We build on a specific idea, originally due to [30]. For any user ρ_i, the "height" of a non-relevant item x_j^- is the number of relevant examples x_k^+ ranked below it by the learned ranking function f_i, i.e.,

$$H_i(x_j^-) = \sum_{k \in L_i^+} \mathbb{1}[f_i(x_k^+) \leq f_i(x_j^-)] , \qquad (1)$$

where $\mathbb{1}[\cdot]$ is the indicator function. The vector $H_i \in \mathbb{R}_+^{n_i^-}$ contains the height of all non-relevant items. A high value of $H_i(x_j^-)$ implies that several relevant items are ranked below the non-relevant x_j^- according to $f_i(\cdot)$. The learning of $f_i(\cdot)$ needs to focus on making the heights of non-relevant items small, i.e., *push* the non-relevant items down, with more emphasis on the large entries in H_i, i.e., non-relevant items towards the top of the list. One can consider learning $f_i(\cdot)$ by minimizing the L_p-norm of H_i [30] (p-norm push), or, the L_∞-norm of H_i (infinite push), which simply focuses on the largest element [3]. Other such suitable functions of H_i, which focus on pushing down non-relevant items from the top of the list, can also be considered.

An alternative approach involves pushing the relevant examples up, rather than pushing the non-relevant examples down. For any user ρ_i, the "reverse height" of a relevant item x_k^+ is the number of non-relevant examples x_j^- ranked above it by the learned ranking function f_i, i.e.,

$$R_i(x_k^+) = \sum_{j \in L_i^-} \mathbb{1}[f_i(x_k^+) \leq f_i(x_j^-)] \ . \tag{2}$$

The vector $R_i \in \mathbb{R}_+^{n_i^+}$ contains the reverse height of all relevant items. A high value of $R_i(x_k^+)$ implies that several non-relevant items are ranked above the relevant x_k^+ according to $f_i(\cdot)$. The learning of $f_i(\cdot)$ needs to focus on making the reverse heights of relevant items small, i.e., *push* the relevant items up. As before, one can choose objective functions on R_i which put more emphasis on relevant items with high reverse heights, e.g., one can use p-norm or infinite push.

The use of indicator functions $\mathbb{1}[f_i(x_k^+) \leq f_i(x_j^-)]$ to define heights, while conceptually convenient, is not suitable for optimization purposes. As a result, we will use a surrogate for the indicator function. For the paper, with$\Delta_i(k,j) = f_i(x_k^+) - f_i(x_j^-)$, we use the logistic loss of the difference as the surrogate:

$$\ell(\Delta_i(k,j)) = \log(1 + \exp(-\Delta_i(k,j))) \ , \tag{3}$$

noting that it forms a convex upper bound to the indicator function. In the sequel, we refer to the height (reverse height) computed using the surrogate, as surrogate height $\tilde{H}_i$ (surrogate reverse height $\tilde{R}_i$).

Collaborative Ranking: Instead of learning the functions $f_i(\cdot)$ separately for each user, we focus on learning them collaboratively. Towards this end, we build on the existing approach of low-rank or latent factor based representation of the scores, widely used in the collaborative filtering literature [25, 39]. Let $U \in \mathbb{R}^{d \times m}$ be the latent factor corresponding to the users, where the i^{th} column $\mathbf{u}_i \in \mathbb{R}^d$ is the latent factor for user ρ_i. Similarly, let $V \in \mathbb{R}^{d \times n}$ be latent factor for the items, where j^{th} column $\mathbf{v}_j \in \mathbb{R}^d$ is the latent factor for item x_j. Then, we assume the scoring function $f_i(x_j) = \mathbf{u}_i^T \mathbf{v}_j$, so that the overall score matrix $F = U^T V \in \mathbb{R}^{m \times n}$ is of rank d, where d is a user specified parameter. Based on such a representation, we have

$$\Delta_i(k,j) = f_i(x_k^+) - f_i(x_j^-) = \mathbf{u}_i^T(\mathbf{v}_k - \mathbf{v}_j) \ . \tag{4}$$

Plugging the above form to compute the surrogate height of a non-relevant sample from (1) gives

$$\tilde{H}_i(x_j^-) = \sum_{k \in L_i^+} \log\Big(1 + \exp(-\mathbf{u}_i^T(\mathbf{v}_k - \mathbf{v}_j))\Big) \ . \tag{5}$$

Similarly, we obtain the form of the surrogate reverse height for a relevant sample from (2) as

$$\tilde{R}_i(x_k^+) = \sum_{j \in L_i^-} \log\Big(1 + \exp(-\mathbf{u}_i^T(\mathbf{v}_k - \mathbf{v}_j))\Big) \ . \tag{6}$$

In the rest of the section, we discuss three different loss functions which work with such surrogate heights and put emphasis on accuracy at the top of the list. We outline algorithms for minimizing such loss functions over the latent factors (U, V).

3.2 P-Norm Push Collaborative Ranking

The main idea in p-norm push collaborative ranking (`P-Push CR`), building on [30], is to consider the L_p norm of the surrogate height vector $\tilde{H}_i$ with elements as in (5), and construct an objective function as a suitable scaled sum of such norms over all users. In particular, we consider the objective

$$\mathcal{E}_{\text{p-push}}(U,V) = \sum_{i=1}^m \frac{\|\tilde{H}_i\|_p^p}{n_i} = \sum_{i=1}^m \frac{1}{n_i} \sum_{j \in L_i^-} \Big(\tilde{H}_i(x_j^-)\Big)^p \tag{7}$$
$$= \sum_{i=1}^m \frac{1}{n_i} \sum_{j \in L_i^-} \left(\sum_{k \in L_i^+} \log\Big(1 + \exp(-\mathbf{u}_i^T(\mathbf{v}_k - \mathbf{v}_j))\Big) \right)^p \ .$$

The optimization is done by alternating minimization [16], i.e., updating U while keeping V fixed, and then updating V while keeping U fixed. The objective function is bi-convex, i.e., convex in one argument while keeping the other fixed. The updates of the latent factors for each user and each item can be done using gradient descent. In each iteration $t+1$:

$$\mathbf{u}_i^{t+1} \leftarrow \mathbf{u}_i^t - \eta \nabla_{\mathbf{u}_i} \mathcal{E}_{\text{p-push}}(U^t, V^t) \ , \quad i = 1, \ldots, m, \tag{8}$$
$$\mathbf{v}_h^{t+1} \leftarrow \mathbf{v}_h^t - \eta \nabla_{\mathbf{v}_h} \mathcal{E}_{\text{p-push}}(U^{t+1}, V^t) \ , \quad h = 1, \ldots, n. \tag{9}$$

The gradients can be obtained by a direct application of chain rule. For compact notation, let

$$\sigma(x) = \frac{1}{1 + \exp(x)} \ . \tag{10}$$

From (3) and (4), we get the gradients

$$\nabla_{\Delta_i(k,j)} \ell(\Delta_i(k,j)) = -\sigma(\Delta_i(k,j)) \ , \tag{11}$$
$$\nabla_{\mathbf{u}_i} \Delta_i(k,j) = (\mathbf{v}_k - \mathbf{v}_j) \ , \tag{12}$$
$$\nabla_{\mathbf{v}_h} \Delta_i(k,j) = \begin{cases} \mathbf{u}_i \ , & \text{if } \ h = k \ , \\ -\mathbf{u}_i \ , & \text{if } \ h = j \ . \end{cases} \tag{13}$$

By chain rule, the gradient of the objective w.r.t. $\mathbf{u}_i$ is

$$\nabla_{\mathbf{u}_i} \mathcal{E}_{\text{p-push}}(U,V)$$
$$= \frac{p}{n_i} \sum_{j \in L_i^-} \left\{ (\tilde{H}_i(x_j^-))^{p-1} \sum_{k \in L_i^+} \sigma(\Delta_i(k,j))(\mathbf{v}_j - \mathbf{v}_k) \right\} .$$

The latent factors $\mathbf{v}_h$ play different roles for different users, depending on whether x_h is relevant or non-relevant for user ρ_i. For item x_h, let Γ_h^+ be the set of users for which x_h is relevant, and letΓ_h^- be the set of users for which x_h is non-relevant. There may be additional users who have no (announced) preference on item x_h, and the optimization of $\mathbf{v}_h$ will not depend on these users. Then, by dividing the sum over all users into sums overΓ_h^+ andΓ_h^- and noting the gradients of$\Delta_i(k,j)$ w.r.t. $\mathbf{v}_h$ for the two cases from (13), by chain rule we have

$$\nabla_{\mathbf{v}_h} \mathcal{E}_{\text{p-push}}(U,V)$$
$$= \sum_{i\in\Gamma_h^-} \frac{p}{n_i} \sum_{j\in L_i^-} \left\{ (\tilde{H}_i(x_j^-))^{p-1} \sum_{k\in L_i^+} \sigma(\Delta_i(k,j))\mathbf{u}_i \right\}$$
$$- \sum_{i\in\Gamma_h^+} \frac{p}{n_i} \sum_{j\in L_i^-} \left\{ (\tilde{H}_i(x_j^-))^{p-1} \sum_{k\in L_i^+} \sigma(\Delta_i(k,j))\mathbf{u}_i \right\} .$$

In practice, instead of summing over all users inΓ_h^+, Γ_h^-, one can pick a mini-batch of users to do the update on, in the flavor of stochastic or mini-batch gradient descent.

3.3 Infinite Push Collaborative Ranking

The main idea in infinite push collaborative ranking (`Inf-Push CR`), building on [3], is to consider the L_∞ norm, i.e., the largest element, of the surrogate height vector $\tilde{H}_i$, and construct an objective function as a suitable scaled sum of such norms over all users. In particular, we consider the objective

$$\mathcal{E}_{\text{inf-push}}(U,V) = \sum_{i=1}^m \frac{\|\tilde{H}_i\|_\infty}{n_i} = \sum_{i=1}^m \frac{1}{n_i} \max_{j\in L_i^-} \tilde{H}_i(x_j^-) \quad (14)$$
$$= \sum_{i=1}^m \frac{1}{n_i} \max_{j\in L_i^-} \left(\sum_{k\in L_i^+} \log\left(1+\exp(-\mathbf{u}_i^T(\mathbf{v}_k - \mathbf{v}_j))\right) \right) .$$

From an optimization perspective, the infinite push objective above is non-smooth, since it considers the maximum over all elements in $\tilde{H}_i$. One can consider a sub-gradient descent approach, or add auxiliary variables to handle the non-smoothness, and possibly consider the dual problem. Such approaches, however, can be slow in practice, and ignore the specific type of non-smoothness in the problem. In our approach we observe that fixing any one variable U or V, the problem is a smooth minimax problem in the other variable, for which efficient gradient-like first order methods exist [26]. We discuss such efficient updates for U, while keeping V fixed, noting that similar updates are possible for V, leading to suitable alternating minimization algorithms.

Consider the latent factor $\mathbf{u}_i$ corresponding to user ρ_i. Assuming V is fixed, the objective function to minimize for $\mathbf{u}_i$ is given by the max-type function

$$f(\mathbf{u}_i) = \max_{j\in L_i^-} f_j(\mathbf{u}_i) \quad (15)$$

where

$$f_j(\mathbf{u}_i) \triangleq \sum_{k\in L_i^+} \log\left(1+\exp(-\mathbf{u}_i^T \mathbf{a}_{kj})\right) , \quad (16)$$

with $\mathbf{a}_{kj} = (\mathbf{v}_k - \mathbf{v}_j)$ being a fixed vector. Note that $f_j(\mathbf{u}_i) = \tilde{H}_i(x_j^-)$, and we are using a more suitable notation for the current discussion illustrating the dependency on $\mathbf{u}_i$. For the development, we assume that each $f_j(\mathbf{u}_i)$ are α-strongly convex and β-smooth [26]. For the current setting, the assumptions are satisfied as long as $\mathbf{u}_i, \mathbf{v}_j$ are bounded.

Following [26], we define the linearization of the max-type function $f(\mathbf{u}_i)$ at $\bar{\mathbf{u}}_i$ as

$$f(\bar{\mathbf{u}}_i; \mathbf{u}_i) = \max_{j\in L_i^-} \left[f_j(\bar{\mathbf{u}}_i) + \langle \nabla f_j(\bar{\mathbf{u}}_i), \mathbf{u}_i - \bar{\mathbf{u}}_i \rangle \right] . \quad (17)$$

In order to define a gradient mapping, we consider a proximal operator of the linearization, given by

$$f_\gamma(\bar{\mathbf{u}}_i; \mathbf{u}_i) = f(\bar{\mathbf{u}}_i; \mathbf{u}_i) + \frac{\gamma}{2}\|\mathbf{u}_i - \bar{\mathbf{u}}_i\|^2 , \quad (18)$$

where $\gamma > 0$. Based on the proximal operator, let

$$f^*(\bar{\mathbf{u}}_i; \gamma) = \min_{\mathbf{u}_i} f_\gamma(\bar{\mathbf{u}}_i; \mathbf{u}_i) , \quad (19)$$
$$\mathbf{u}_i^*(\bar{\mathbf{u}}_i; \gamma) = \operatorname*{argmin}_{\mathbf{u}_i} f_\gamma(\bar{\mathbf{u}}_i; \mathbf{u}_i) , \quad (20)$$
$$\mathbf{g}_f(\bar{\mathbf{u}}_i; \gamma) = \gamma(\bar{\mathbf{u}}_i - \mathbf{u}_i^*(\bar{\mathbf{u}}_i; \gamma)) , \quad (21)$$

where $\mathbf{g}_f(\bar{\mathbf{u}}_i; \gamma)$ is the gradient mapping of the max-type function f at $\bar{\mathbf{u}}_i$ [26, Section 2.3]. Assuming the individual functions f_j, the gradient method for the minimax problem is given by the simple iterative update:

$$\mathbf{u}_i^{t+1} \leftarrow \mathbf{u}_i^t - \eta \mathbf{g}_f(\mathbf{u}_i^t; \gamma) , \quad (22)$$

where η is the learning rate and $\mathbf{u}_i^t$ serves as the linearization point $\bar{\mathbf{u}}_i$. Since each function $f_j(\mathbf{u}_i)$ is β-smooth, we can set $\gamma = \beta$. With any $\eta \le \frac{1}{\beta}$, the gradient method has an exponential convergence rate [26, Theorem 2.3.4], which is substantially better than just treating the problem as non-smooth optimization and using sub-gradient descent [26].

The key step in obtaining the gradient mapping in (21) is to solve the optimization problem in (19). Using the definition of the linearization and the proximal operator [27], we get the following optimization problem:

$$\min_{\mathbf{u}_i} \left\{ \max_{j\in L_i^-} [f_j(\bar{\mathbf{u}}_i) + \langle \nabla f_j(\bar{\mathbf{u}}_i), \mathbf{u}_i - \bar{\mathbf{u}}_i \rangle] + \frac{\gamma}{2}\|\mathbf{u}_i - \bar{\mathbf{u}}_i\|^2 \right\} . \quad (23)$$

The above problem is the proximal operator of a piecewise linear function. The problem can be rewritten as a constrained optimization problem:

$$\begin{aligned} &\min_{\mathbf{u}_i, t} \ t + \frac{\gamma}{2}\|\mathbf{u}_i - \bar{\mathbf{u}}_i\|^2 \\ &\text{s.t.} \quad f_j(\bar{\mathbf{u}}_i) + \langle \nabla f_j(\bar{\mathbf{u}}_i), \mathbf{u}_i - \bar{\mathbf{u}}_i \rangle \le t . \end{aligned} \quad (24)$$

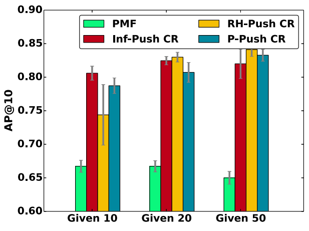
A direct calculation shows that the dual of the constrained problem is equivalent to a quadratic minimization problem over the probability simplex:

$$\min_{\mathbf{y}} \mathbf{y}^T Q \mathbf{y} + \mathbf{b}^T \mathbf{y} \quad \text{s.t.} \quad \sum_j y_j = 1, \quad y_j \ge 0 , \quad (25)$$

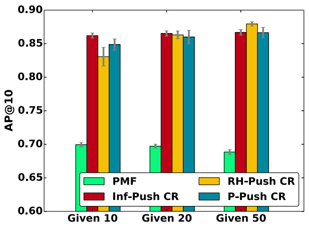
where Q is a positive semi-definite matrix. The dual objective is concave, and simply the negative of the above quadratic form, and $\mathbf{y}$ corresponds to the Lagrange multipliers. The quadratic problem with simplex constraints can be solved by a projected gradient or exponentiated gradient method [15, 18]. In practice, we use early stopping of the dual projected gradient descent, compute the gradient mapping, and update $\mathbf{u}_i$ based on (22).

3.4 Reverse-Height Push Collaborative Ranking

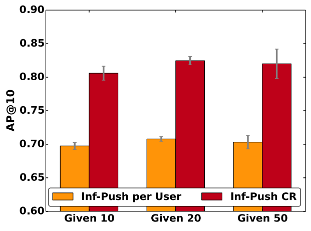
We now consider a formulation based on the reverse height $R_i(x_k^+)$ of positive items x_k^+ for user ρ_i as in (2). As discussed in [30], the reverse height is similar to the rank of a positive item x_k^+, but only considering the negative items x_j^- which are above it, and disregarding the other positive items. By treating reverse height as a surrogate for rank, one can follow the existing literature on learning to rank, and consider maximizing objectives such as reciprocal rank or discounted cumulative gain (DCG), e.g., for each user ρ_i,



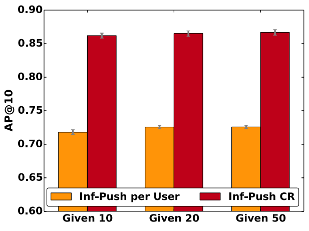
(a) MovieLens 100K



(b) MovieLens 1M



(c) MovieLens 100K



(d) MovieLens 1M

Figure 1: (a),(b): Ranking vs. Rating Prediction (PMF) for the two MovieLens datasets. All Push CR methods outperform PMF in terms of AP@10. (c),(d): `Inf-Push CR` vs. Infinite Push per User for the two MovieLens datasets. Collaboratively learning ranking function results in better top 10 performance than constructing different ranking model per user.

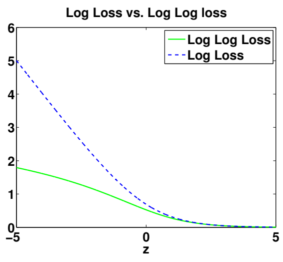
maximize

$$\sum_{k \in L_i^+} \frac{1}{R_i(x_k^+)} \quad \text{or} \quad \sum_{k \in L_i^+} \frac{1}{\log(1 + R_i(x_k^+))} .$$

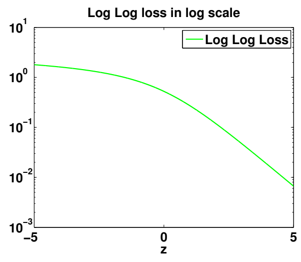
However, these objectives are difficult to directly maximize. Instead, staying in the spirit of these objectives, we consider the following objective

$$\sum_{k \in L_i^+} \log \left(\frac{1}{1 + R_i(x_k^+)} \right) ,$$

which decreases with increasing rank.



(a) Log-log vs. logistic loss



(b) Log-log loss in log-scale

Figure 2: (a) A comparison of log-log loss with the logistic loss, and (b) the log-log loss with a y-log scale.

To maintain consistency with the earlier formulations, we consider minimizing the negative of the objective. Further, using the surrogate reverse height $\tilde{R}_i(x_k^+)$ as in (6), and considering the objective over all users, we focus on minimizing the following objective for reverse height push collaborative ranking:

$$\mathcal{E}_{\text{rh-push}}(U, V) = \sum_{i=1}^{m} \frac{1}{n_i} \sum_{k \in L_i^+} \log(1 + \tilde{R}_i(x_k^+)) \qquad (26)$$

where

$$\tilde{R}_i(x_k^+) = \sum_{j \in L_i^-} \log \left(1 + \exp(-\mathbf{u}_i^T(\mathbf{v}_k - \mathbf{v}_j)) \right) . \qquad (27)$$

Denoting$\Delta_i(k, j) = z$ for simplicity, the above loss has components of the form $\log(1 + \log(1 + \exp(-z))$, which we refer to as log-log loss. Interestingly, while the log-log loss is not convex, it is quasi-convex and monotonic decreasing, as shown in Figure 2. Further, the objective is more robust than the logistic loss for negative z, and stays closer to the 0-1 loss for which it acts as a surrogate. Related non-convex losses have been explored in the literature, e.g., in the context of t-logistic regression [14].

As in Section 3.2, we use alternating minimization of the objective in (26), i.e., updating U keeping V fixed, and updating V keeping U fixed. In the current setting, the alternating minimization will reach a stationary point, possibly a local minimum of the objective. As we discuss in Section 4, the empirical results from the log-log loss are surprisingly good along with fairly stable parameter sensitivity, which implies the need for further study of the loss. For alternating minimization, the updates can be done using gradient descent:

$$\mathbf{u}_i^{t+1} \leftarrow \mathbf{u}_i^t - \eta \nabla_{\mathbf{u}_i} \mathcal{E}_{\text{rh-push}}(U^t, V^t) , \quad i = 1, \ldots, m, \qquad (28)$$

$$\mathbf{v}_h^{t+1} \leftarrow \mathbf{v}_h^t - \eta \nabla_{\mathbf{v}_h} \mathcal{E}_{\text{rh-push}}(U^{t+1}, V^t) , \quad h = 1, \ldots, n. \qquad (29)$$

As before, the gradients can be obtained by chain rule. Using (11) and (12), from (26) we have

$$\nabla_{\mathbf{u}_i} \mathcal{E}_{\text{rh-push}}(U, V)$$
$$= \frac{1}{n_i} \sum_{k \in L_i^+} \left\{ \frac{1}{1 + \tilde{R}_i(x_k^+)} \sum_{j \in L_i^-} \sigma(\Delta_i(k, j))(\mathbf{v}_j - \mathbf{v}_k) \right\} .$$

Recall that the latent factors $\mathbf{v}_h$ play different roles for different users, depending on whether x_h is relevant or not for user ρ_i. As before, for item x_h, let Γ_h^+ be the set of users for which x_h is relevant, and letΓ_h^- be the set of users for which x_h is non-relevant. There may be additional users who have no (announced) preference on item x_h, and the optimization of $\mathbf{v}_h$ will not depend on these users. Then, by dividing the sum over all users into sums overΓ_h^+ and Γ_h^- and noting the gradients of$\Delta_i(k, j)$ w.r.t. $\mathbf{v}_h$ for the two cases from (13), from (26), by chain rule we have

$$\nabla_{\mathbf{v}_h} \mathcal{E}_{\text{rh-push}}(U, V)$$
$$= \sum_{i \in \Gamma_h^-} \frac{1}{n_i} \sum_{j \in L_i^-} \left\{ \frac{1}{1 + \tilde{R}_i(x_k^+)} \sum_{k \in L_i^+} \sigma(\Delta_i(k, j)) \mathbf{u}_i \right\}$$
$$- \sum_{i \in \Gamma_h^+} \frac{1}{n_i} \sum_{j \in L_i^-} \left\{ \frac{1}{1 + \tilde{R}_i(x_k^+)} \sum_{k \in L_i^+} \sigma(\Delta_i(k, j)) \mathbf{u}_i \right\} .$$

Again, in practice, instead of summing over all users in Γ_h^+, Γ_h^-, one can pick a mini-batch of users to do the update on, in the flavor of stochastic or mini-batch gradient descent.

Remark: For all of our proposed formulations we consider an additional regularization term $\frac{\lambda}{2}(\|U\|^2 + \|V\|^2)$. The gradient term for the $\mathbf{u}_i$ updates then has an additional term $\lambda\mathbf{u}_i$, and similarly for the $\mathbf{v}_h$ updates. Also, the computational complexity for computing the gradients for a user ρ_i in one iteration is $\mathcal{O}(n_i^+ n_i^-)$, which is small in practice since $n_i^+, n_i^- \ll n$, the total number of items.

4. EXPERIMENTS

We conducted extensive experiments in order to validate the performance of the three proposed collaborative ranking approaches (`P-Push CR`, `Inf-Push CR`, `RH-Push CR`) on real world datasets. We have divided our experiments into three sets in order to address the following questions: (i) What is the impact of posing the problem as a rating prediction problem versus a ranking one, i.e., optimizing for the square loss versus a ranking loss with focus on the top of the list? (ii) Is learning the ranking functions *collaboratively* across all users more effective than constructing a separate ranking model per user? (iii) Do the proposed push collaborative ranking approaches outperform state-of-the-art CR approaches for recommendation systems? After discussing the experimental setup, we evaluate these questions empirically in Sections 4.2, 4.3, and 4.4 respectively.

4.1 Experimental Setting

Datasets. We used four publicly available datasets for our experiments: three movie datasets MovieLens 100K, MovieLens 1M, EachMovie and a musical artist dataset from Yahoo! which contains user ratings of music artists [41]. The two MovieLens datasets were kept in their original form. The EachMovie dataset used was a subset of the original EachMovie including the ratings of 30,000 randomly selected users with 20 or more ratings[1]. The Yahoo! dataset was subsampled to speed up the experiments, since the original dataset had close to 2 million users and 100,000 items. For the Yahoo! dataset, we first selected the 1,000 most popular items and then selected the 6,000 most heavy users (with the most ratings). In addition to the subsampling we removed the user ratings of the special value 255 ('never play again') and rescaled user ratings from 0-100 to the 1-5 interval. The rescaling was done by mapping 0-19 to 1, 20-39 to 2, etc. The procedure followed for the Yahoo! dataset is close to the one described in [38]. The users, items and ratings statistics are summarized in Table 1.

Dataset	# Users	# Items	# Ratings
MovieLens 100K	943	1,682	100,000
MovieLens 1M	6,040	3,706	1,000,209
EachMovie	30,000	1,623	2,109,780
Yahoo! R1	6,000	1,000	3,522,232

Table 1: Dataset Statistics

Data Split. Throughout our experiments we adopted the "weak" generalization setting which has become standard practice in the literature [5, 22, 38, 39]. In this setting, we fix the number of training ratings per user $n_i = \{10, 20, 50\}$ and randomly choose n_i ratings for each user for training, 10 ratings for validation, and test on the remaining ratings. The experimental setting facilitates the comparison with the state-of-the-art approaches and allows the study of the ranking algorithms' sensitivity to the number of available training ratings per user. Users with less than $n_i + 20$ ratings are removed to ensure that we can evaluate on at least 10 ratings for each user. Note that the number of test items vary significantly across users, with many users having many more test ratings than training ones, thus simulating the real life recommendation scenario [38]. For each setting of 'given n_i' ratings per user we repeated the whole procedure 10 times and reported the mean and standard deviation of the results.

[1] http://mlcomp.org/datasets/350

Setup. In the evaluation phase, for each user, we compare pairs of items $(x_h, x_{h'})$ belonging to the test set for that user, such that each of these items have appeared in the training set of some users. Using such items ensures that the latent factors of the items have been updated during training, and we are not working with their randomly initialized value. In addition, in order to simulate the setup of two-level relevance ratings (Section 3), we transformed the ratings of all datasets from a multi-level relevance scale to a two-level scale $(+1, -1)$, i.e., using 4 as relevance threshold for MovieLens 100K, MovieLens 1M and Yahoo!, and 5 for EachMovie. In particular, for the two MovieLens datasets and the Yahoo! dataset, ratings of the value 4-5 are marked as relevant, and 1-3 are marked as non-relevant. For the EachMovie dataset 5-6 are marked as relevant, and 1-4 are marked as non-relevant. Also, we removed users with no positive (relevant) or no negative (non-relevant) ratings in the training set, so that there is at least one positive example to "pull up" or at least one negative example to "push down". The exact same input was given to all algorithms.

Evaluation Metrics. Since our goal is to achieve accuracy towards the top of the ranked list, we chose two evaluation metrics which reflect this and have been widely used in the learning to rank literature [7, 11, 21].

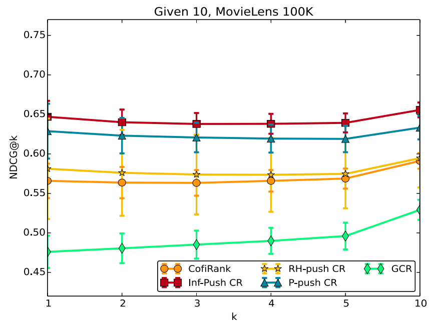
In the definitions below, we assume that the items are enumerated according to the decreasing order of their scores. We will denote with k the truncation threshold, reflecting the number of recommendations we want to consider. Let r be the index ranging over positions in the ranked list ($r = \{1, \ldots, k\}$), and $rel(r)$ denote the relevance of the r-th item for the user. For the definition of *Average Precision@k*, $rel(r) = 0$ if this is a non-relevant item for the user, $rel(r) = 1$ otherwise. For the computation of *NDCG@k*, $rel(r)$ equals to the user's observed rating from the multi-level scale, i.e., 1-6 for EachMovie, 1-5 for the rest of the datasets. The latter choice was made for the sake of consistency with the state-of-the-art literature [38, 39], though during training all approaches have access just to the two-level relevance scale ratings.

Average Precision@k (*AP@k*) is defined as the average of Precisions computed at each relevant position in the top k items of the user's ranked list. *Precision@r* (*P@r*) equals the fraction of relevant items out of the top r ranked items.

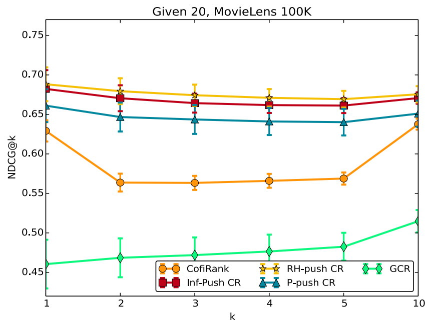
$$AP@k = \sum_{r=1}^{k} \frac{P@r \cdot rel(r)}{\min(k, \#\text{ of relevant items})} \tag{30}$$

DCG@k captures the importance of finding the correct ordering among higher ranked items compared to that among lower ranked items, by discounting the importance weight with the item's position in the ranked list. *DCG@k* is formally given by

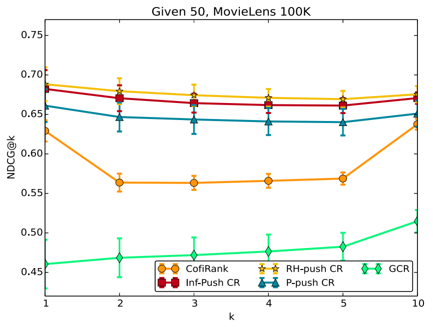
$$DCG@k = \sum_{r=1}^{k} \frac{2^{rel(r)} - 1}{\log_2(r+1)} \tag{31}$$



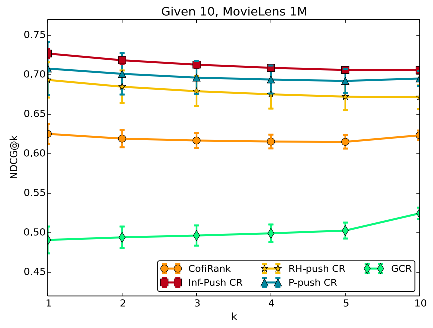
(a) MovieLens 100K, Given10



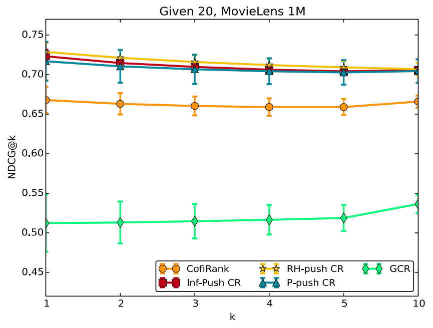
(b) MovieLens 100K, Given20



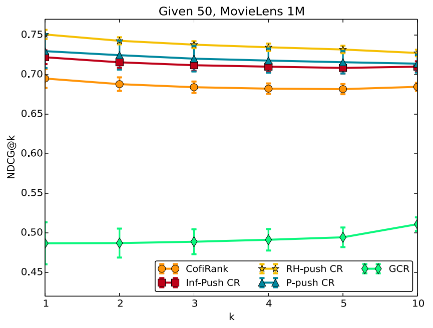
(c) MovieLens 100K, Given50



(d) MovieLens 1M, Given10

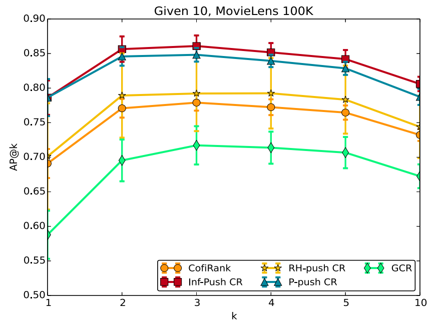


(e) MovieLens 1M, Given20

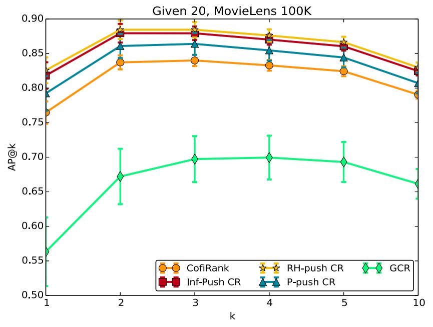


(f) MovieLens 1M, Given50

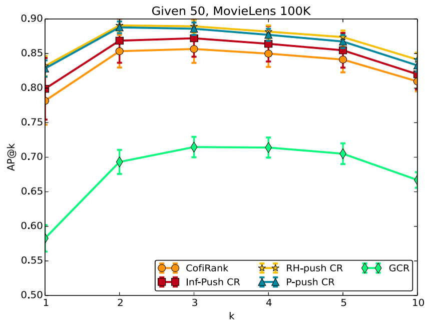
Figure 3: Comparison of performance of the proposed Push CR models against CofiRank and GCR in NDCG@k for $k = \{1, 2, 3, 4, 5, 10\}$ for the two MovieLens datasets. All of our proposed Push CR methods outperform CofiRank and GCR.



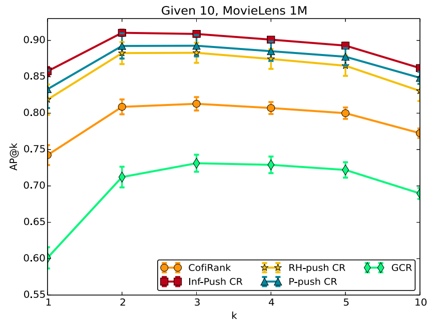
(a) MovieLens 100K, Given10



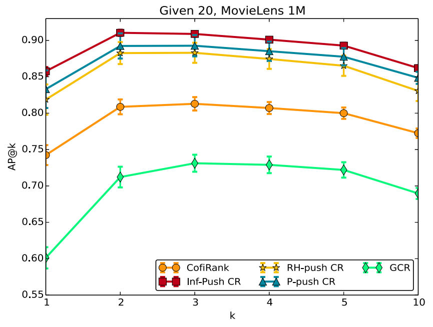
(b) MovieLens 100K, Given20



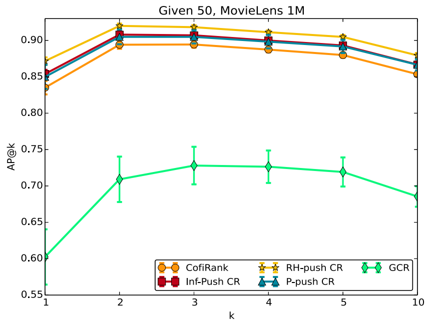
(c) MovieLens 100K, Given50



(d) MovieLens 1M, Given10

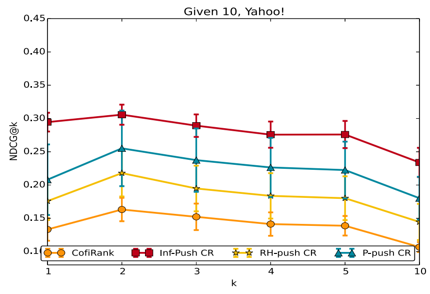


(e) MovieLens 1M, Given20

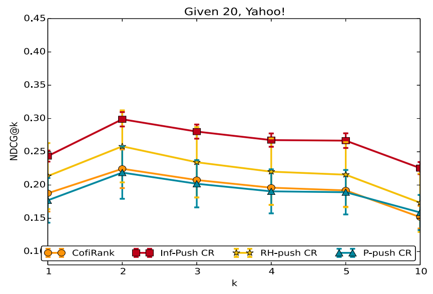


(f) MovieLens 1M, Given50

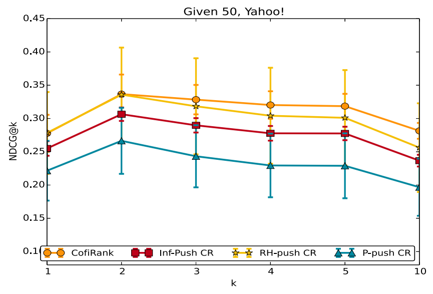
Figure 4: Comparison of performance of the proposed Push CR models against CofiRank and GCR in Average Precision@k for $k = \{1, 2, 3, 4, 5, 10\}$ for the two MovieLens datasets. All of our proposed Push CR methods outperform CofiRank and GCR.



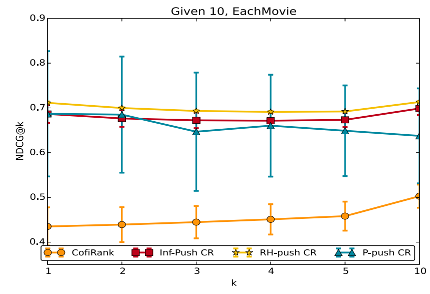
(a) Yahoo!, Given10



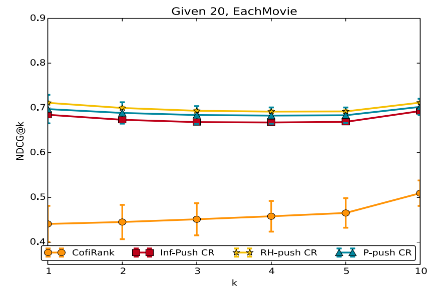
(b) Yahoo!, Given20



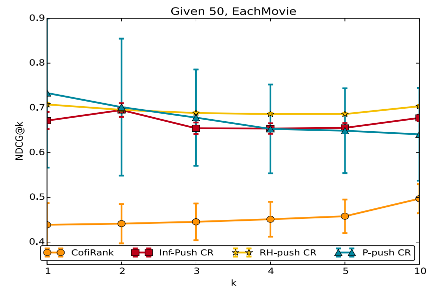
(c) Yahoo!, Given50



(d) EachMovie, Given10



(e) EachMovie, Given20



(f) EachMovie, Given50

Figure 5: Comparison of performance of the proposed Push CR models vs. CofiRank in NDCG@k for the Yahoo! and EachMovie datasets. In most cases at least one (usually all) of our proposed Push CR methods outperforms CofiRank.

$NDCG@k$ is the ratio of $DCG@k$ to the Ideal $DCG@k$ for that user, i.e the $DCG@k$ obtained had the user's recommendation list been sorted according to the ground truth ratings given by the user.

In our experiments, after computing the metrics per user, we averaged the results over all users and reported the results for Mean Average Precision@k (MAP@k) and Mean NDCG@k. We varied k from 1 to 10 as this is usually the size of a recommendation list fitting a device's screen.

Model Parameters. The validation set was used for selecting the best model parameters. We varied the regularization coefficient λ in the range $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1\}$ and the rank of the latent factors in {5, 10, 20, 30, 50, 70, 90}. We fixed the learning rate for the gradient based updates to be $\eta = 10^{-4}$ for MovieLens 1M and Yahoo!, and $\eta = 10^{-3}$ for MovieLens 100K and EachMovie. In addition, the validation set was used to determine early stopping. We stopped either after 200 iterations or when the improvement in MAP@5 in the validation set got smaller than some threshold (10^{-4}).

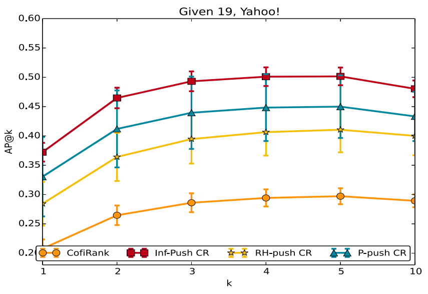
For the `p-Push CR` method we set the parameter $p = 2$, which gives a small push towards the top of the list. The reason we chose a small p was because of numerical instabilities which appeared for higher values of p. For the inner optimization problem (23) of the `Inf-Push CR` approach, we used the following parameter setting: We set γ of the proximal operator equal to 10, learning rate equal to 0.01 and regularization coefficient $1/\gamma = 0.1$. Early stopping was used, since there was no need to solve the inner problem fully as in the next outer iteration the same optimization problem would have to be solved. We stopped either when the improvement in the value of the optimization problem was smaller than 0.01 or after 25 iterations.

4.2 Validation: Ranking vs. Rating Prediction

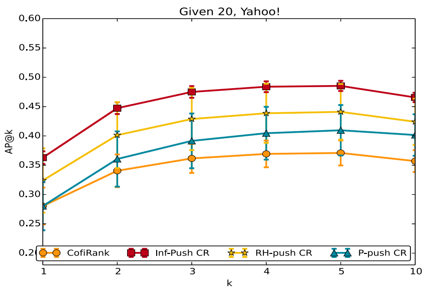
In the first set of experiments, we validate the effect of optimizing for a ranking based loss function versus the square loss. Since such a comparison has been explored in the recent literature [5, 39], we limit the set of experiments to a direct comparison between Probabilistic Matrix Factorization (PMF) [25] (*rating prediction*) and the proposed Push CR approaches (*ranking*) in terms of AP@10. For the experiments, we used the two MovieLens datasets, with the number n_i of training ratings per user set to vary in the range of $\{10, 20, 50\}$. Figures 1(a) and 1(b) illustrate the superiority of optimizing a ranking based loss function compared to the square loss. The results support the effectiveness of posing recommendation systems as a ranking problem rather than a rating prediction problem based on squared loss. Thus, in the sequel we focus on comparing ranking approaches.

4.3 Validation: Collaborative vs. Individual

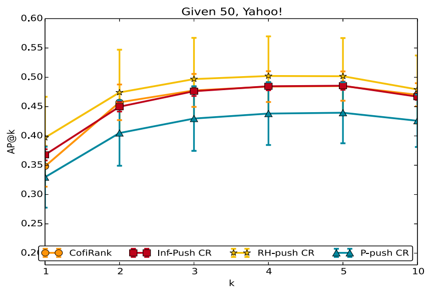
The second set of experiments is conducted to examine whether collaborative ranking, i.e., learning latent factors for the users and items from the rating matrix, results in higher ranking accuracy compared to training separate ranking functions per user. For this task, we compare the proposed collaborative model for Infinite Push (`Inf-Push CR`) with the approach of building a separate Infinite Push model per user (Inf-Push per User). We again used the two MovieLens datasets, varying the number n_i of training ratings per user in the range of $\{10, 20, 50\}$ and reported the mean and standard deviation of AP@10. For Inf-Push per User we implemented Infinite Push as in [3] and used the feature vectors of the items (movie genre) in order to learn the linear ranking function [1, 33]. Note that according to [33], genre seems to be the most informative feature among other types



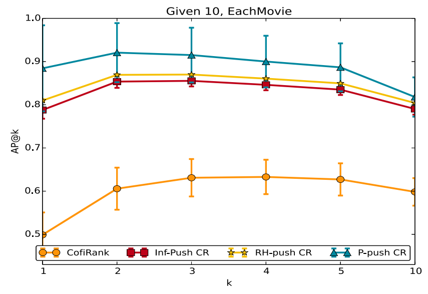
(a) Yahoo!, Given10



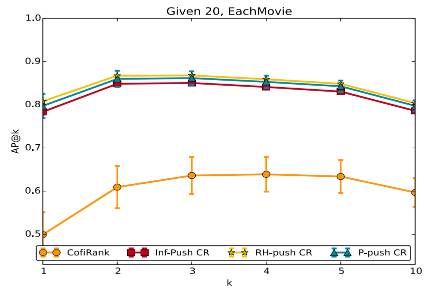
(b) Yahoo!, Given20



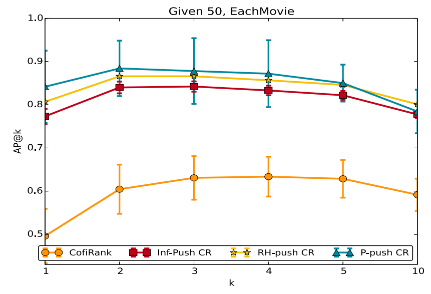
(c) Yahoo!, Given50



(d) EachMovie, Given10



(e) EachMovie, Given20



(f) EachMovie, Given50

Figure 6: Comparison of performance of the proposed Push CR models vs. CofiRank in AP@k for the Yahoo! and EachMovie datasets. In all cases at least one (usually all) of our proposed Push CR methods outperforms CofiRank.

of side information. Further, any helpful feature one can add for the items, the collaborative model can also be extended to take advantage of such features. Figures 1(c) and 1(d) show that learning the latent factors collaboratively clearly outperforms learning a separate ranking function per user using the rated items' features. The experiment validates the effectiveness of the collaborative ranking framework and allows us to focus on CR approaches for recommendation for the rest of the paper.

4.4 Comparison of Proposed Approaches

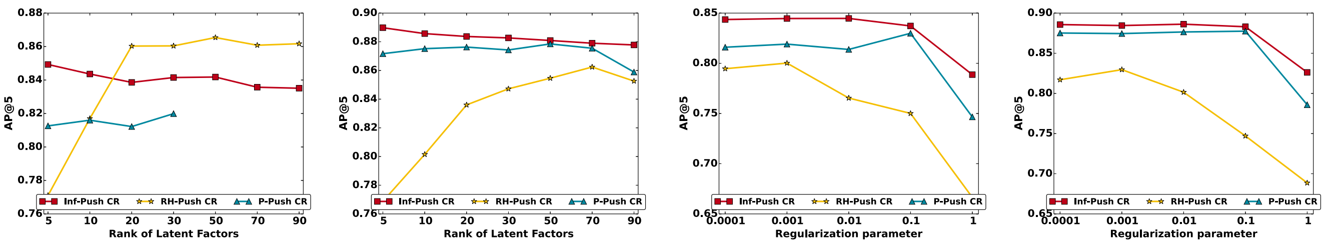
In the third set of experiments, we address the question: Do our three proposed CR approaches, i.e., `p-Push CR`, `Inf-Push CR`, `RH-Push CR` outperform the state-of-the-art CR methods? We compare the Push CR methods with CofiRank [39], which is considered a very popular baseline in the CR literature and with Global Collaborative Ranking (GCR) [22] which is the ranking counterpart of PMF-type methods. For CofiRank, we used the publicly available package[2] and set the same parameter values provided in the original paper [39] (rank = 100 and $\lambda = 10$), and default values provided in the source code for unstated parameters. For GCR, we implemented the source code based on the publicly available package[3] since the original code used slightly different experimental setting than ours, making the comparison of the different methods difficult. In the GCR experiments, we chose the local model rank in the validation set, varying it in the range $\{5, 10, 15, 20\}$ and the regularization parameter λ in $\{10^{-3}, 10^{-2}, 10^{-1}\}$, and kept the rest of the parameters in their default values given in the package. Also, we optimized for the average number of mis-ordered pairs (0-1 error) [22, Eq (6)] and we stopped the algorithm either when the improvement in the 0-1 error in the validation set got smaller than some threshold (0.001) or after 200 iterations.

[2] www.cofirank.org [3] prea.gatech.edu

Results and Discussion. We report the mean and standard deviation of AP@top and NDCG@top over 10 runs, where 'top' varies in the range of {1, 2, 3, 4, 5, 10} and the fixed number of training ratings per user (n_i) varies in {10, 20, 50}. Recall that the parameters, such as rank, regularization parameter, and early stopping, for the Push CR methods were all chosen based on performance on AP@5 in the validation set. Figures 3(a)-(c), 3(d)-(f), 5(a)-(c), 5(d)-(f) compare the competing methods in terms of NDCG@top for the MovieLens 100K, 1M, EachMovie and Yahoo! datasets respectively, whereas figures 4(a)-(c), 4(d)-(f), 6(a)-(c), 6(d)-(f) show the corresponding AP@top comparison.

Based on the experimental results illustrated in Figures 3, 4, 5, 6 we conclude that in most cases at least one (usually all) of our proposed Push CR methods outperforms CofiRank in terms of both ranking metrics. The only case where the mean of NDCG@top of CofiRank outperforms the proposed approaches is for Yahoo!, Given 50. For the same case, the mean of AP@top of CofiRank almost coincides with that of `Inf-Push CR`, is higher than that of `P-Push CR` but is outperformed by `RH-Push CR`. For the case of Yahoo!, Given 20, the NDCG@ top of CofiRank outperforms that of `P-Push CR` but lags behind `Inf-Push CR` and `RH-Push CR`. So, overall, our Push CR approaches significantly outperform CofiRank. In addition, from our experiments in the two MovieLens datasets, we observe that CofiRank and all proposed Push CR methods always outperform GCR in terms of the two top-k ranking metrics.

We can also compare the performance among the proposed Push CR methods. The improvement of `Inf-Push CR`



(a) MovieLens 100K (b) MovieLens 1M (c) MovieLens 100K (d) MovieLens 1M

Figure 7: Effect of Rank of Latent Factors and Regularization parameter on AP@5 for MovieLens 100K and MovieLens 1M, for $n_i = 10$. (a)(b) The rank varies in $\{5, 10, 20, 30, 50, 70, 90\}$ while the regularizer is fixed to 0.0001. (c)(d) The regularizer λ varies in $\{0.001, 0.001, 0.1, 1\}$ while rank= 10. For high values of rank and λ, AP@5 decreases.

is particularly notable when the amount of available training points (n_i) is small. For n_i=10, `Inf-Push CR` outperforms all other methods in terms of both AP@top and NDCG@top, in all datasets apart from EachMovie (where it lags behind `P-Push CR`). In contrast, the improvement of `RH-Push CR` is particularly notable when the amount of available training points (n_i) grows larger. For n_i=50, `RH-Push CR` outperforms the rest of Push CR methods in both AP@top and NDCG@top, except for EachMovie, where the mean of AP@top lags behind that of `P-Push CR`. Also, `P-Push CR` has the most variability in its performance across the ten runs.

Comparison with CLiMF

For completeness, we also compare the proposed Push CR methods with another state-of-the-art method, CLiMF [34]. Such a comparison is tricky as CLiMF uses implicit binary ratings data while Push CR methods use explicit binary ratings. We train CLiMF on the explicit feedback datasets by binarizing the rating values using 4 as the relevance threshold, treating ratings of value 1-3 as 0 (not observed). Otherwise, we evaluate CLiMF using exactly the same setup as in the rest of the methods. For this experiment, we use the MovieLens 100K and 1M datasets, fix the number of training ratings per user $n_i = 20$, choose best parameters (λ: $\{10^{-4}$, 10^{-3}, 10^{-2}, 10^{-1},1$\}$, rank: $\{5, 10, 20, 30, 50, 70, 90\}$) in the validation set and report the mean of AP@5 and NDCG@5 over ten random runs. Results are reported in Table 2 and show that the Push CR methods are competitive, usually better on AP@5 and NDCG@5 compared to CLiMF. Similar trends were observed for different values of 'top' and for $n_i = 10$, $n_i = 50$, and are thus omitted.

	MovieLens 100K		MovieLens 1M	
Method	AP@5	NDCG@5	AP@5	NDCG@5
CLiMF	0.8446	0.6424	0.8787	0.6982
`Inf-Push CR`	0.8605	0.6612	**0.8938**	0.7042
`RH-Push CR`	**0.8665**	**0.6693**	0.8926	**0.7092**
`P-Push CR`	0.8443	0.6402	0.8865	0.7028

Table 2: Comparison of `Push CR` methods with CLiMF.

4.5 Parameter Sensitivity Analysis

Figures 7(a)-(d) show how the Push CR models behave with different combinations of parameters. We used the two MovieLens datasets, fixing the number of training ratings per user n_i=10. To study the effect of the rank of the latent factors on the model performance, we varied it in the range of $\{5, 10, 20, 30, 50, 70, 90\}$ while keeping the regularizer λ fixed to 0.0001. Figures 7(a)-(b) show that there is a decrease in AP@5 for high values of rank (70, 90). For `P-Push CR`, in the MovieLens 100K dataset, higher values of rank resulted in numerical instability issues. Also, in order to study the effect of the regularizer on the model performance, we varied λ in the range of $\{0.0001, 0.001, 0.01, 1\}$ while keeping the rank fixed to 10. Figures 7(c)-(d) show that for all three methods there is a decrease in AP@5 for high values of λ (0.1, 1). Note that we have performed early stopping based on the validation set, which leads to robustness of our methods to different parameter choices.

5. CONCLUSION

The paper builds on and merges two keys themes from the collaborative filtering and learning to rank literature: the use of a low rank representation for collaborative scoring, and the use of ranking losses which focus on the accuracy at the top of the list, by pushing down non-relevant items and/or pulling up relevant items. The merge results in a family of novel collaborative ranking algorithms with a push at the top of the ranked list for each user. The proposed algorithms are simple and scalable from an optimization perspective and are shown to be competitive to state-of-the-art baselines through extensive experiments. The modular structure of the develoment shows that other representations, beyond low rank, and other ranking losses, beyond the three considered here, can be considered with simple changes to the ideas presented.

Our methods are applicable in a variety of settings where (i) explicit binary relevance ratings are available (i.e., helpfulness of a review, like/dislike of a song) and (ii) implicit binary ratings are treated as explicit (i.e., considering non-clicks as non-relevant [13], using proxies such as dwell time as a threshold to quantify item relevance [42]). However, we plan to extend Push CR methods to handle multi-level relevance ratings as well. Future work will also consider developing parallel versions of the updates as well as incorporating feature information from users and items when available.

Acknowledgments. We thank the anonymous reviewers for the detailed comments. We thank Suju Rajan and her team at Yahoo! for helpful discussions related to the paper. The work was supported in part by NSF grants IIS-1447566, IIS-1422557,CCF-1451986, CNS-1314560, IIS-0953274, IIS-1029711, NASA grant NNX12AQ39A, and a gift from Yahoo!. We also acknowledge technical support from the University of Minnesota Supercomputing Institute.

6. REFERENCES

[1] D. Agarwal and B.-C. Chen. Regression-based latent factor models. In *KDD*, 19–28, 2009.

[2] D. Agarwal, B.-C. Chen, P. Elango and R. Ramakrishnan. Content recommendation on web portals. In *Communications of the ACM*, 56(6): 92–101, 2013.

[3] S. Agarwal. The infinite push: A new support vector ranking algorithm that directly optimizes accuracy at the absolute top of the list. In *SDM*, 839–850, 2011.

[4] R. Baeza-Yates and R-N. Berthier. Modern information retrieval. In *ACM*, (463), 1999.

[5] S. Balakrishnan and S. Chopra. Collaborative ranking. In *WSDM*, 143–152, 2012.

[6] R. M. Bell and Y. Koren. Lessons from the netflix prize challenge. In *ACM SIGKDD Explorations Newsletter*, 9(2): 75–79, 2007.

[7] S. Boyd, C. Cortes, M. Mohri, and A. Radovanovic. Accuracy at the top. In *NIPS*, 953–961, 2012.

[8] C. JC Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton and G. Hullender. Learning to rank using gradient descent. In *ICML*, 89–96, 2005.

[9] C. JC Burges. From RankNet to LambdaRank to LambdaMART: An overview. In *Learning*, 11: 23–581, 2010.

[10] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *ICML*, 129–136, 2007.

[11] O. Chapelle and Y. Chang. Yahoo! Learning to Rank Challenge Overview. In *Yahoo! Learning to Rank Challenge*, 1–24, 2011.

[12] K. Crammer and Y. Singer. Pranking with ranking. In *NIPS*, 14: 641–647, 2011.

[13] A. S Das, M. Datar, A. Garg and S. Rajaram. Google news personalization: scalable online collaborative filtering. In *WWW*, 271–280, 2007.

[14] N. Ding, SVN Vishwanathan, M. Warmuth and V. S. Denchev. T-logistic regression for binary and multiclass classification. In *JMLR*, (5): 1–55, 2013.

[15] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra. Efficient projections onto the l1-ball for learning in high dimensions. In *ICML*, 272–279, 2008.

[16] P. Jain, P. Netrapalli, Y. Singer, and S. Sanghavi. Low-rank matrix completion using alternating minimization. In *ACM symposium on Theory of computing*, 665–674, 2013.

[17] K. Järvelin and J. Kekäläinen. IR evaluation methods for retrieving highly relevant documents In *SIGIR*, 41–48, 2000.

[18] J. Kivinen and M. K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132(1): 1–63, 1997.

[19] Y. Koren, R. Bell and C. Volinsky. Matrix factorization techniques for recommender systems. In *Computer IEEE*, 42(8): 30–37, 2009.

[20] Y. Koren and J. Sill. Ordrec: an ordinal model for predicting personalized item rating distributions. In *RecSys*, 117–124, 2011.

[21] Q. Le, and A. Smola. Direct optimization of ranking measures. In *arXiv preprint arXiv:0704.3359*, 2007.

[22] J. Lee, S. Bengio, S. Kim, G. Lebanon, and Y. Singer. Local collaborative ranking. In *WWW*, 85–96, 2014.

[23] G. Linden, B. Smith and J. York. Amazon. com recommendations: Item-to-item collaborative filtering. In *Internet Computing, IEEE*, 7(1): 76–80,2003.

[24] T.-Y. Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3): 225–331, 2009.

[25] A. Mnih and R. Salakhutdinov. Probabilistic matrix factorization. In *NIPS*, 1257–1264, 2007.

[26] Y. Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer, 2004.

[27] N. Parikh and S. Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3): 123–231, 2013

[28] A. Rakotomamonjy. Sparse support vector infinite push. *arXiv preprint arXiv:1206.6432*, 2012.

[29] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. BPR: Bayesian personalized ranking from implicit feedback. In *UAI*, 452–461, 2009.

[30] C. Rudin. The p-norm push: A simple convex ranking algorithm that concentrates at the top of the list. In *JMLR*, 10: 2233–2271, 2009.

[31] R. Salakhutdinov and A. Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *ICML*, 880–887, 2008.

[32] D. Sculley. Large scale learning to rank. In *NIPS Workshop on Advances in Ranking*, 1–6, 2009.

[33] H. Shan, and A. Banerjee. Generalized probabilistic matrix factorizations for collaborative filtering. In *ICDM*, 1025–1030, 2010.

[34] Y. Shi, A. Karatzoglou, L. Baltrunas, M. Larson, N. Oliver, and A. Hanjalic. CLIMF: learning to maximize reciprocal rank with collaborative less-is-more filtering. In *RecSys*, 139–146, 2012.

[35] Y. Shi, A. Karatzoglou, L. Baltrunas, M. Larson, and A. Hanjalic. GAPfm: Optimal top-n recommendations for graded relevance domains. In *CIKM*, 2261–2266, 2013.

[36] Y. Shi, A. Karatzoglou, L. Baltrunas, M. Larson, and A. Hanjalic. xCLIMF: optimizing expected reciprocal rank for data with multiple levels of relevance. In *RecSys*, 431–434, 2013.

[37] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. In *JMLR*, 1453–1484, 2005.

[38] M. Volkovs and R. S. Zemel. Collaborative ranking with 17 parameters. In *NIPS*, 2294–2302, 2012.

[39] M. Weimer, A. Karatzoglou, Q. V. Le, and A. Smola. CofiRank: Maximum margin matrix factorization for collaborative ranking. In *NIPS*, 2007.

[40] J. Xu and H. Li. Adarank: a boosting algorithm for information retrieval. In *SIGIR*, 391–398, 2007.

[41] Yahoo! R1 dataset `webscope.sandbox.yahoo.com`

[42] X. Yi, L. Hong, E. Zhong, N.N. Liu and S. Rajan. Beyond Clicks: Dwell Time for Personalization. In *RecSys*, 2014.