Local Collaborative Ranking

Joonseok Lee
Georgia Tech
Atlanta, GA, USA
jlee716@gatech.edu

Samy Bengio
Google Research
Mountain View, CA, USA
bengio@google.com

Seungyeon Kim
Georgia Tech
Atlanta, GA, USA
seungyeon.kim@gatech.edu

Guy Lebanon
Amazon
Seattle WA, USA
glebanon@gmail.com

Yoram Singer
Google Research
Mountain View, CA, USA
singer@google.com

ABSTRACT

Personalized recommendation systems are used in a wide variety of applications such as electronic commerce, social networks, web search, and more. Collaborative filtering approaches to recommendation systems typically assume that the rating matrix (e.g., movie ratings by viewers) is low-rank. In this paper, we examine an alternative approach in which the rating matrix is *locally low-rank*. Concretely, we assume that the rating matrix is low-rank within certain neighborhoods of the metric space defined by (user, item) pairs. We combine a recent approach for local low-rank approximation based on the Frobenius norm with a general empirical risk minimization for ranking losses. Our experiments indicate that the combination of a mixture of local low-rank matrices each of which was trained to minimize a ranking loss outperforms many of the currently used state-of-the-art recommendation systems. Moreover, our method is easy to parallelize, making it a viable approach for large scale real-world rank-based recommendation systems.

Categories and Subject Descriptors

[**Information retrieval**]: Retrieval tasks and goals—*Recommender systems*; [**Information systems applications**]: Data mining—*Collaborative filtering*; [**Machine Learning**]: Supervised learning—*Ranking*

Keywords

recommender systems, collaborative filtering, ranking

1. INTRODUCTION

Collaborative filtering is a popular approach in recommendation systems whose goal is to estimate the rating of a user u for an item i based on a partial set of (user, item) ratings. The set of ratings can be viewed as a partially observed matrix $M \in \mathbb{R}^{m\times n}$, with m users and n items. In order to recommend new items, we form predictions for unobserved entries in M, turning the recommendation task into a matrix completion task.

Incomplete SVD is a popular matrix completion setting that uses squared loss and assumes that M is low rank:

$$(\hat{U},\hat{V}) = \operatorname*{arg\,min}_{U,V} \sum_{(u,i)\in \mathtt{A}} \left(M_{u,i} - [UV^T]_{u,i}\right)^2, \qquad (1)$$

where $U \in \mathbb{R}^{m\times r}$, $V \in \mathbb{R}^{n\times r}$ (r is substantially smaller than both m and n), and $\mathtt{A}$ is the training set of observed entries in M. The estimated matrix $\hat{M} = \hat{U}\hat{V}^\top$ can be used to estimate ratings $M_{v,j}$ of user v and item j, including user-item pairs that are not in the training set $(v, j) \notin \mathtt{A}$. After the matrix M is completed, a recommendation for a particular user v is formed by picking the items corresponding to the k highest values in $\{M_{v,j} : (v, j) \notin \mathtt{A}\}$.

Despite its popularity, the incomplete SVD approach often performs poorly compared to more recent models based on ranked loss minimization. The main reason is that the SVD in (1) focuses on the task of accurately predicting the actual values of unobserved entries of M, while ranked loss minimization methods focus on accurately picking the top k recommended items.

In contrast to standard collaborative filtering models, our method does not assume that the matrix M of user-item ratings is low-rank. Instead, we follow [25] and assume that M is locally low-rank where locality is defined by a neighborhood with respect to a given metric on pairs of (row, column) indices. In contrast to [25] that uses squared loss reconstruction, we consider a ranked loss minimization setting that is more appropriate for real world recommendation systems. The resulting method, Local Collaborative Ranking (LCR), outperforms both standard ranked collaborative filtering models, the model proposed in [25], as well as more traditional CF methods. Furthermore, due to the locality assumption, our approach is highly parallelizable and scales up to large scale datasets.

We start by reviewing the rating prediction problem and the approach that is used to solve it in Section 2. We then formalize the ranking problem in Section 3. In Section 4, we describe our model in detail followed by experimental analysis in Section 5. We then review related work in Section 6 and summarize contribution and future work in Section 7.

WWW'14, April 7–11, 2014, Seoul, Korea.
ACM 978-1-4503-2744-2/14/04.
Include the http://DOI string/url which is specific for your submission and included in the ACM rightsreview confirmation email upon completing your IW3C2 form.

2. LOW-RANK MATRIX APPROXIMATION

We recall here the notations from the previous section: the matrix $M \in \mathbb{R}^{m\times n}$ denotes the matrix of user-item ratings, and the observed training data is $\{(u, i, M_{u,i}) : (u, i) \in \mathtt{A}\}$ where $\mathtt{A}$ is the set of user-item training ratings.

2.1 Global SVD

Low rank matrix approximation attempts to find a low-rank matrix $\hat{M}$ that best approximates M. Since $\hat{M}$ is low-rank, it can be written as $\hat{M} = \hat{U}\hat{V}^T$ where $\hat{U} \in \mathbb{R}^{m\times r}$, $\hat{V} \in \mathbb{R}^{n\times r}$ with $r \ll \min(m, n)$. The most well known algorithm to do so is incomplete SVD (1), which minimizes the squared reconstruction error over the training set. The resulting approximation $\hat{M} = \hat{U}\hat{V}^T$ is then used to estimate ratings and suggest recommendations. Method (1) is one of the most effective approaches for rating prediction in recommender systems [20, 27] and has been extensively studied in the machine learning literature (see for example [24, 35, 34, 21, 33, 26]). The objective function in (1) is non-convex and an iterative method such as alternating least square (ALS) or stochastic gradient descent (SGD) should converge to a local minimum. Often, regularization terms $\Omega(U) = \sum_{u,k} U^2_{u,k}$ and $\Omega(V) = \sum_{i,k} V^2_{i,k}$ are added to the objective function in (1) to avoid overfitting.

2.2 LLORMA

The Locally Low Rank Matrix Approximation (LLORMA) model [25] assumes that the space of (row,column) pairs $\Phi = \{(u, i) : u = 1, \ldots, m, i = 1 \ldots, n\}$ is endowed with a distance function d that measures distances between pairs of users and items. The distance function leads to the notion of neighborhoods of user-item pairs, and local low-rank assumption states that M can be approximated by a set of low-rank matrices $\hat{M}_s$, $s \in \Phi$, where each $\hat{M}_s$ is low rank and approximates M particularly well in the neighborhood of s: $N_s = \{s' \in \Phi : d(s, s') < \alpha\}$. Thus, M is approximated by a number of low-rank matrices – one for each neighborhood – and each of these matrices describes the original matrix for some subset of users and items. Note that the distance function $d((a, b), (c, d))$ needs not be small when $|c - a|$ and $|d - c|$ are small since the assignment of users and items to rows and columns is arbitrary. Thus, the neighborhoods N_s do not correspond to neat square or circular patches in the array corresponding to the matrix M.

The intuition behind the LLORMA assumption is as follows: the entire rating matrix M is not low-rank but a submatrix M_s, restricted to certain types of similar users and items (for example, children users viewing cartoon movies) is low-rank. We refer the reader to [25] for information on estimating the distance d from the training data. We proceed with a formal description of the LLORMA model.

LLORMA first identifies q neighborhoods surrounding q anchor points $s_1, \ldots, s_q \in \Phi$. In the experiments we sampled the anchor points uniformly from the training set, but it is conceivable that more sophisticated adaptive techniques will result in a more accurate model. Then, it estimates a low-rank approximation for each neighborhood by minimizing the squared reconstruction error, weighted by the proximity of the reconstruction site to the anchor point. Formally, each local model is learned by

$$\arg\min_{U,V} \sum_{(u,i)\in\mathtt{A}} K((u^*, i^*), (u, i)) \left([UV^T]_{u,i} - M_{u,i}\right)^2, \quad (2)$$

where $K((u^*, i^*), (u, i))$ is a two-dimensional smoothing kernel that measures the proximity of the reconstruction site (u, i) to the anchor point (u^*, i^*). This kernel function may be defined in several ways. LLORMA implements this using a product of two smoothing kernels $K(u^*, u)\ K(i^*, i)$, one for users and the other for items. The smoothing kernels are inversely related to distance function, for example $K(x, y) = \exp(-cd(x, y))$ (Gaussian kernel) or $K(x, y) = (1 - d(x, y)/h)I(d(x, y) < h)$ (triangular kernel). The optimization problem above is essentially a weighted version of the global SVD optimization problem, but it needs to be repeated q times - once for each anchor point.

Unfortunately, it is computationally impractical to solve a new weighted SVD problem above for all user-item prediction pairs. Thus, instead of treating each test user-item pair as an anchor point and solving the corresponding model (2), the anchor points are selected before the test user-item pairs are observed. The q anchor points lead to q local models that are then linearly combined to provide the estimate of the test user-item rating. The specific linear combination rule is given by locally constant regression or non-parametric Nadaraya-Watson regression

$$\hat{M}_{u,i} = \sum_{t=1}^{q} \frac{K((u_t, i_t), (u, i))}{\sum_{s=1}^{q} K((u_s, i_s), (u, i))} [\hat{U}_t\hat{V}_t^T]_{u,i}, \quad (3)$$

where (u_t, i_t) is the anchor point of local model t. In other words, (3) is a convex combination of each local model's prediction, ensuring that points closer to the queried point (u, i) contribute more than those far from it. More details on locally constant regression and other forms of non-parametric regression may be found in any book on non-parametric statistical modeling, for example [41].

The problem with global SVD that is mentioned in the introduction section applies also to LLORMA: both global and local SVD focus on squared error minimization and are thus suitable for rating prediction. However, the primary use case of modern recommendation systems is in producing a ranked list of recommendations, and it is the relevancy of the top items on that list that is crucial to the success of the recommendation system. In this paper, we derive an analog of ranked loss analog of LLORMA.

3. LABEL RANKING

A general label ranking problem is defined as follows. Let X be a domain of instances and Y be a set of labels, possibly of infinite cardinality. A *label ranking* for an instance $x \in X$ is a total ordering over Y, where $y_1 \succ y_2$ implies that y_1 is preferred to y_2 for x. A *label ranking function* $f : X \times Y \to \mathbb{R}$ returns a score indicating how much a label y is relevant to an instance x. We learn f so as to preserve preference order as much as possible on training data. That is, given x, for all $y_1 \succ y_2$ relationship, we fit f to satisfy $f(x, y_1) > f(x, y_2)$ (if possible).

In the context of recommendation systems, X is a set of users $\mathcal{U}$ and Y is a set of items $\mathcal{I}$. The function $f(u, i)$ estimates a preference score for a user $u \in \mathcal{U}$ on an item $i \in \mathcal{I}$. We fit f such that $f(u, i_1) > f(u, i_2)$ if user u prefers item i_1 to item i_2.

One way to learn this ranking function is to sort the output of a rating predictor using the approach in Section 2. With this approach, f directly approximate $f(u,i) \approx M_{u,i}$ for all (u,i) pairs on the training set $\mathtt{A}$. With this ranking function f, the ordered sequence of $f(u,1),\ldots,f(u,n)$ gives the total order over the labels. (Recall that n is the number of items.) This approach is called a "point-wise" method.

Although it is straightforward to learn to approximate ratings with point-wise techniques, doing so suffers from calibration problem [15], or in other words the interpretation of scores for one user may not be consistent with the interpretation of the same scores by another user. Of course, for both users an ordered pair of items means the same thing: one item is preferred to another. Directly approximating rating values (rather than the ordering of the ratings) misses this effect and may thus lead to lower accuracy.

The "pair-wise" approach avoids these drawbacks by considering the preference order seen in training data, rather than directly estimating the values themselves. With this approach, $f(u,i)$ does not necessarily fit to $M_{u,i}$. Instead, it tries to preserve the relative order of preferences between two ratings from the same user. That is, it minimizes a risk function

$$\mathcal{E}(f) = \sum_{u\in\mathcal{U}} \sum_{i,j\in M_u} \mathcal{L}(f(u,i) - f(u,j), M_{u,i} - M_{u,j}) \tag{4}$$

where M_u is the set of items rated by the user u. One choice of loss function $\mathcal{L}(x,y)$ is the zero-one loss, penalizing only when the signs of x and y disagrees. A differentiable approximation for this function will be introduced later.

One drawback of the pair-wise method is its computational complexity. As the pair-wise method sums over all pairs of ratings by the same user, its worst-case complexity is $O(k^2)$ for k ratings or per rating average of $O(k^2/m)$. Obviously, it takes much longer than a point-wise method, whose complexity is $O(k)$. Fortunately, however, rating matrices are usually very sparse, so the number of pairs may not be intractably large. Also, since many recommendation ratings lie in a finite set of scores (for example one to five stars), many ties are bound to occur and dropping these ties leads to manageable complexity.

4. COLLABORATIVE RANKING

We now turn our attention to the main thrust of the paper where we present our learning-to-rank algorithm by minimizing a pair-wise loss function. With a pair-wise loss, we are not interested in the absolute ratings but rather in the *difference* in preference values. Let (i,j) denote the indices of two items rated by a user u. Without loss of generality we assume that item i is preferred over item j, namely $M_{u,i} > M_{u,j}$. The training set for pair-wise losses is constructed using observed entries from $\mathtt{A}$ as follows

$$\{(u,i,j,M_{u,i} - M_{u,j}) : (u,i),(u,j) \in \mathtt{A}, M_{u,i} > M_{u,j}\}. \tag{5}$$

We use the same factorization form UV^T for our model with $U \in \mathbb{R}^{m\times r}$ and $V \in \mathbb{R}^{n\times r}$ where $r \ll \min(m,n)$. In contrast to the single rating approach presented in Section 2, $\hat{M}_{u,i}$ would not necessarily be an approximation to $M_{u,i}$ in value. That is, for a pair $M_{u,i} > M_{u,j}$, our low-rank estimation should conform with the relative order and difference in values of $\hat{M}_{u,i}$ and $\hat{M}_{u,j}$.

In the sequel, we discuss pair-based loss functions. Then, we describe a natural adaptation of global SVD from Section 2.1 to pair-wise loss functions. Finally, we describe the *Local Collaborative Ranking (LCR)* method.

4.1 Pair-wise loss functions

A natural loss we consider in ranking is the rate of preference disagreement, namely, how many pairs are mis-ordered by the ranking model. We refer to this loss as the zero-one error for pairs,

$$\mathcal{E}(f) = \sum_{u\in\mathcal{U}} \frac{1}{s_u} \sum_{k=1}^{s_u} \mathbf{1}[\Delta M_k \cdot \Delta f(x_k) < 0], \tag{6}$$

where x_k designates an ordered pair rated by user u. ΔM_k and $\Delta f(x_k)$ denote the difference in observed and predicted rating for the pair. s_u is the number of ordered items rated by the user u. We normalize each user's pair disagreement number by s_u so that each user is weighed equally. The operator $\mathbf{1}$ designates the indicator function, $\mathbf{1}[\mathtt{true}] = 1$ and $\mathbf{1}[\mathtt{false}] = 0$. The error described by (6) amounts to the average number of pairs for which the predicted preference disagrees with the observed one and can be viewed as an instance of the generalized ranking loss in [9].

The zero-one loss (6) is not differentiable and finding an f that minimizes it is computationally intractable. Instead, we use a smooth surrogate loss function $\mathcal{L}$ that forms a convex upper bound on the zero-one loss function

$$\mathcal{E}(f) = \sum_{u\in\mathcal{U}} \frac{1}{s_u} \sum_{k=1}^{s_u} \mathcal{L}(\Delta M_k, \Delta f(x_k)). \tag{7}$$

We describe and experiment with two different families of margin-based loss functions, provided in Table 1. The first scales the loss by ΔM, while the second constructs an additive margin using ΔM. These losses are analogous to similar relaxations of the zero-one loss in classification.

4.2 A Global Approximation

Similarly to the incomplete SVD in Section 2.1, we can learn a global model by minimizing the loss described in (7). The minimization produces two matrices U, V that when multiplied UV^T they provide an estimate for ratings whose ordering conform to the ordering in the training data. To simplify our derivation we define a ternary function g as follows:

$$\begin{aligned} g(u,i,j) &\stackrel{def}{=} f(u,i) - f(u,j) \\ &= [UV^T]_{u,i} - [UV^T]_{u,j} \\ &= \sum_{k=1}^{r} U_{u,k}V_{i,k} - \sum_{k=1}^{r} U_{u,k}V_{j,k}. \end{aligned} \tag{8}$$

Substituting g in (7) we get

$$\mathcal{E}(U,V) = \sum_{u\in\mathcal{U}} \frac{1}{s_u} \sum_{(i,j)\in M_u} \mathcal{L}(\Delta M_{u,i,j}, g(u,i,j)), \tag{9}$$

where M_u is the list of items rated by the user u and $\Delta M_{u,i,j} = M_{u,i} - M_{u,j}$. Each pair (i,j) is considered once when $M_{u,i} > M_{u,j}$. Since $\Delta M_{u,i,j}$ does not depend on U and V it is viewed as a constant in the optimization process. U and V are fitted by gradient-based updates:

$$U_{u,k} \leftarrow U_{u,k} - \nu \frac{\partial \mathcal{E}(U,V)}{\partial U_{u,k}}, \quad V_{i,k} \leftarrow V_{i,k} - \nu \frac{\partial \mathcal{E}(U,V)}{\partial V_{i,k}}$$

where ν is the learning rate.

Loss	Multiplicative version [M]	Additive version [A]
Log-loss	$\mathcal{L}_{\text{Log[M]}}(\Delta M, \Delta f) = \Delta M \log(1 + \exp\{\gamma - \Delta f\})$	$\mathcal{L}_{\text{Log[A]}}(\Delta M, \Delta f) = \log(1 + \exp\{\gamma + \Delta M - \Delta f\})$
Exp-loss	$\mathcal{L}_{\text{Exp[M]}}(\Delta M, \Delta f) = \Delta M \exp\{\gamma - \Delta f\}$	$\mathcal{L}_{\text{Exp[A]}}(\Delta M, \Delta f) = \exp\{\gamma + \Delta M - \Delta f\}$
Hinge-loss	$\mathcal{L}_{\text{Hinge[M]}}(\Delta M, \Delta f) = \Delta M[\gamma - \Delta f]_+$	$\mathcal{L}_{\text{Hinge[A]}}(\Delta M, \Delta f) = [\gamma + \Delta M - \Delta f]_+$

Table 1: Pair-based loss functions. The scalar γ is a free parameter which designate a target margin value.

We calculate below the derivative of $\mathcal{E}(U, V)$ with respect to the entries of U and V using the chain rule. To obtain the partial derivatives with respect to $U_{u,k}$, we sum over all pairs of two items i and j rated by the user u. In this case, we can still assume without loss of generality that $M_{u,i} > M_{u,j}$. When calculating the partial derivatives with respect to $V_{i,k}$ we note that there are two possible cases: item i is preferred to j or vice versa. The two cases contribute to the two inner summations in (11).

$$\frac{\partial \mathcal{E}(U,V)}{\partial U_{u,k}} = \frac{1}{s_u} \sum_{(i,j) \in M_u} \frac{\partial L(\Delta M_{u,i,j}, g)}{\partial g(u,i,j)} \frac{\partial g(u,i,j)}{\partial U_{u,k}} \tag{10}$$

$$\frac{\partial \mathcal{E}(U,V)}{\partial V_{i,k}} = \sum_{u \in \mathcal{U}} \frac{1}{s_u} \left[\sum_{j: M_{u,i} > M_{u,j}} \frac{\partial \mathcal{L}(\Delta M_{u,i,j}, g)}{\partial g(u,i,j)} \frac{\partial g(u,i,j)}{\partial V_{i,k}} + \sum_{j: M_{u,i} < M_{u,j}} \frac{\partial \mathcal{L}(\Delta M_{u,i,j}, g)}{\partial g(u,i,j)} \frac{\partial g(u,i,j)}{\partial V_{j,k}} \right]. \tag{11}$$

The partial derivatives of g are given by

$$\frac{\partial g(u,i,j)}{\partial U_{u,k}} = V_{i,k} - V_{j,k} \tag{12}$$

$$\frac{\partial g(u,i,j)}{\partial V_{a,k}} = \begin{cases} U_{u,k} & \text{if } a = i \\ -U_{u,k} & \text{if } a = j \end{cases}. \tag{13}$$

The last step is the calculation of the partial derivatives of the loss functions in Table 1 with respect to g. For example in the case of the log-loss we have

$$\frac{\partial \mathcal{L}_{\text{Log[M]}}(\Delta M, g)}{\partial g} = \frac{\partial}{\partial g} \Delta M \log(1 + e^{\gamma - g}) = \frac{-\Delta M e^{\gamma - g}}{1 + e^{\gamma - g}}.$$

The partial derivatives need to be adjusted if regularization is used, for example L_2 regularization $\Omega(U) = \sum_{u,k} U_{u,k}^2$ and $\Omega(V) = \sum_{i,k} V_{i,k}^2$ adds a simple linear term to the partial derivatives.

4.3 Local Collaborative Ranking (LCR)

In LCR, we apply the ideas of local low-rank matrix approximation to the ranking loss minimization framework. Each local low-rank model covers a different subgroup of users and items (see Section 2), which are combined in prediction time using locally constant regression.

Extending LLORMA to ranking loss minimization is not straightforward as LLORMA uses two kernel functions (one for users $K(u_t, u)$ and the other for items $K(i_t, i)$) as described in Section 2.2 while pair-wise losses are formed based on a single user and two items in each term $g(u, i, j)$. This leads to three kernel functions: $K(u_t, u), K(i_t, i)$, and $K(i_t, j)$.

To estimate the local models, we minimize the sum of ranking losses $\mathcal{L}$ (as in (9)) for all possible item pairs of the same user. Formally, we need to solve

$$\arg\min_{U_t, V_t} \sum_{u \in \mathcal{U}} \frac{1}{s_u} \sum_{(i,j) \in M_u} \mathcal{L}(\Delta M_{u,i,j}, g(u,i,j)) \tag{14}$$

where $t = 1, \ldots, q$ and $\Delta M_{u,i,j} = M_{u,i} - M_{u,j}$. The main difference from (9) is the definition of $g(u, i, j)$. Instead of the difference of predictions of a single global model ($f(u, i) - f(u, j)$), it is now a combination of local models:

$$\begin{aligned} g(u,i,j) &\equiv \hat{M}_{u,i} - \hat{M}_{u,j} \\ &= \sum_{t=1}^{q} \frac{K((u_t,i_t),(u,i))}{\sum_{s=1}^{q} K((u_s,i_s),(u,i))} [U_t V_t^T]_{u,i} \\ &\quad - \sum_{t=1}^{q} \frac{K((u_t,i_t),(u,j))}{\sum_{s=1}^{q} K((u_s,i_s),(u,j))} [U_t V_t^T]_{u,j}. \end{aligned} \tag{15}$$

After estimating the local models, we combine them at test time using locally constant regression (as in the case of LLORMA)

$$\hat{M}_{u,i} = \sum_{t=1}^{q} \frac{K((u_t,i_t),(u,i))}{\sum_{s=1}^{q} K((u_s,i_s),(u,i))} [U_t V_t^T]_{u,i}, \tag{16}$$

where (u_t, i_t) is the anchor point of local model t. Note that this amounts to a convex combination of local models, weighted by the proximity of the local anchor point to the predicted user-item pair.

We can calculate the gradients similarly to (10) and (11), so long as the loss function $\mathcal{L}$ is differentiable. Specifically, we have

$$\frac{\partial g(u,i,j)}{\partial [U_t]_{u,k}} = [V_t]_{i,k} - [V_t]_{j,k} \tag{17}$$

$$\frac{\partial g(u,i,j)}{\partial [V_t]_{a,k}} = \begin{cases} [U_t]_{u,k} & \text{if } a = i \\ -[U_t]_{u,k} & \text{if } a = j \end{cases}. \tag{18}$$

4.3.1 Parallelism

Though we did not explicitly present the full description of the gradient calculation, it is evident that it is no longer possible to parallelize the gradient computation as in the case of LLORMA. In LLORMA, each local model does not interact with other local models during training, so each local model can be learned in parallel (asynchronously). However, it is difficult to break the objective function (14) with (15) into independent optimization problems since the estimate of each local model influences the other local models.

Nevertheless, we can still take advantage of parallelism in a slightly different and more complex way. As long as all local models are at the same phase of the learning process, we only need the current global estimation $\hat{M}_{u,i}$ and not the estimation of each individual local model. Since the global estimate is the same for every local model so long as we establish a *synchronization* mechanism, we can compute combined predictions at the beginning of each iteration, freeze the values, and let each local model use these values during the gradient estimation. We can thus still update each local model in parallel, as long as the current estimation $\hat{M}_{u,i}$ is synchronized at each iteration.

To recap, the learning process consists of the following steps: we first initialize each model at random; then, we repeatedly alternate between the following two steps until a convergence criterion is met: (i) we calculate current estimations of each local model *(Estimation step)*, and (ii) we update each local model concurrently using the estimations *(Update step)*. While the estimation step requires synchronization, the update step can be done asynchronously. Since the update step is typically the most computationally expensive, the above computational scheme can significantly reduce run time.

4.3.2 Distance and Smoothing Kernel

In order to complete the derivation of the algorithm, we need to endow the space of recommendations with a metric between users and analogously a metric between items. Such metrics can be constructed using side information, for example, users' age, gender, and so on (either using standard distances or using metric learning techniques, for example [45] or [22, 23, 10]) However, many datasets (including most public datasets) do not include such data, in which case d can be based on the partially observed matrix M.

Algorithm 1 The LCR Learning Algorithm

Input: $M \in \mathbb{R}^{m \times n}$
Parameters: number of local models q, local rank r, learning rate ν, regularization coefficient λ
Define: A as the set of observed entries in M
for all $t \in \{1, ..., q\}$ **do**
  Initialize $U_t \in \mathbb{R}^{m \times r}, V_t \in \mathbb{R}^{n \times r}$ randomly.
  Pick an observed pair (u_t, i_t) from M at random.
end for
while not-converged **do**
  // *Estimation step (Synchronization)*
  for all $(u, i) \in$ A **do**
    $w_{u,i} \leftarrow \sum_{t=1}^{q} K(u_t, u) K(i_t, i)$
    $f_{u,i} = \sum_{t=1}^{q} \frac{K(u_t,u)K(i_t,i)}{w_{u,i}} \left[U_t V_t^T\right]_{u,i}$
  end for
  for all $(u, i), (u, j) \in$ A **do**
    $\ell_{u,i,j} = \left.\frac{\partial E}{\partial g}\right|_{g=f_{u,i}-f_{u,j}}$
  end for
  // *Update step*
  for all $t \in \{1, ..., q\}$ **in parallel do**
    $\forall u \in \{1, ..., m\} : [\Delta U]_u \leftarrow 0$
    $\forall i \in \{1, ..., n\} : [\Delta V]_i \leftarrow 0$
    for all $(u, i) \in$ A and $(u, j) \in$ A **do**
      if $M_{u,i} > M_{u,j}$ **then**
        $[\Delta U]_u \leftarrow [\Delta U]_u + (\frac{K(u_t,u) \cdot K(i_t,i)}{w_{u,i}} \cdot [V_t]_i$
        $- \frac{K(u_t,u) \cdot K(i_t,j)}{w_{u,j}} \cdot [V_t]_j) \cdot \ell_{u,i,j}$
        $[\Delta V]_i \leftarrow [\Delta V]_i + \frac{K(u_t,u) \cdot K(i_t,i)}{w_{u,i}} \cdot [U_t]_u \cdot \ell_{u,i,j}$
        $[\Delta V]_j \leftarrow [\Delta V]_j - \frac{K(u_t,u) \cdot K(i_t,j)}{w_{u,j}} \cdot [U_t]_u \cdot \ell_{u,j,i}$
      end if
    end for
    $\forall u \in \{1, ..., m\} : \ [U_t]_u \leftarrow [U_t]_u - \nu(\frac{[\Delta U]_u}{|\mathcal{U}| \cdot s_u} + \lambda [U_t]_u)$
    $\forall i \in \{1, ..., n\} : \ [V_t]_i \leftarrow [V_t]_i - \nu(\frac{[\Delta V]_i}{|\mathcal{U}| \cdot s_u} + \lambda [V_t]_i)$
  end for
end while
output: $U_t V_t^T, \ t = 1, \ldots, q$

Algorithm 2 The LCR Prediction

Input: learned local models $U_t V_t^T (t = 1, \ldots, q)$, test point (u^*, i^*)
return $\hat{M}_{u,i} = \sum_{t=1}^{q} \frac{K(u_t,u^*)K(i_t,i^*)}{\sum_{s=1}^{q} K(u_s,u^*)K(i_s,i^*)} \left[U_t V_t^T\right]_{u^*,i^*}$

One possible approach is to construct the distance between two users based on the correlation of the two users (excluding unobserved ratings), and construct similarly the distance between two items. This approach does not perform well since the partially observed matrix is very sparse, leading to poor estimates of correlation and distance. Instead, we factorize M using incomplete SVD (1) and obtain two latent representation matrices U and V of users and items respectively. We then proceed to construct the distance between two users based on the normalized inner-product of row i and row j of the matrix U,

$$\arccos\left(\frac{\langle u_i, u_j \rangle}{\|u_i\| \cdot \|u_j\|}\right),$$

where we denote by u_i and u_j the i and j rows of U. The analogous expression with V replacing U may be used to construct the distance between two items.

We used a product kernel $K((u_t, i_t), (u, i)) = K(u_t, u)K(i_t, i)$, where each component kernel is an Epanechnikov kernel ($K(s_1, s_2) = 3/4(1 - d(s_1, s_2)^2)\mathbf{1}[d(s_1, s_2) < h]$) with distances computed as described above. We also tried uniform and triangular kernel, but the performance was worse than Epanechnikov kernel, in agreement with the theory of kernel smoothing [41].

4.3.3 The Algorithm

The pseudo-code of the resulting learning algorithm is provided in Algorithm 1. We use an L_2 regularization scheme, but other kinds of regularization can be used instead. Prediction of an unseen test example (u^*, i^*) is described in Algorithm 2.

5. EXPERIMENTS

We conducted experiments to evaluate the performance of the proposed LCR approach on real world datasets. In the first set of experiments, we varied the LCR hyper-parameters in order to better understand the dependency of LCR on them. In the second set of experiments, we compared the performance of LCR method with other well-known ranking methods for recommendation systems.

5.1 Experimental Setting

Data Split.

We evaluate the performance of the learning algorithm by separating the data into a training part, used to train the model, and a test part, used to evaluate it. In order to evaluate the sensitivity of the algorithm to the size of the training set, one can vary either the number of available training ratings per user, or the proportion of training ratings per user, keeping the rest of the ratings for the test set. Since users have a variable number of ratings [38, 32], the number of training and test ratings per user can vary significantly depending on this choice.

In the literature, CofiRank [43] introduced an experimental setting which fixes the number of training ratings per user. In this setting, we randomly choose a constant number N of training ratings per user, and all other ratings are kept in the test set. Users without enough ratings are dropped from the test set. For example, for a user with 100 ratings and $N = 10$, we use 10 ratings for training and 90 for test. Another user with 8 ratings is dropped as this user does not satisfy the minimum requirement. This evaluation procedure has low variance since users in the test set typically have a large number of ratings. The evaluation is biased towards frequent raters, as we drop users with few ratings. This evaluation method has recently became standard practice [43, 1, 40, 12] and thus we adopt it in our experiments where we compare the performance of LCR with others methods (Section 5.3).

The alternative evaluation method keeps a fixed proportion of ratings for each user in the training set and moves the rest of the observations to the test set (regardless of the number of available ratings for each user). For example, if the ratio is 0.5, a user with 100 ratings will have 50 of its ratings kept for training and the remaining 50 used for testing. A user with 8 ratings is not dropped, but trained with 4 ratings and tested with the remaining 4 ratings. This scheme is less biased since it contains users with a few ratings as well as users with many ratings, but it may have higher variance due to the inclusion of users with a small number of ratings.

We experiment with both approaches. In the first set of experiments (Section 5.2) where we investigate the dependency of LCR on its hyper-parameters, we used the fixed ratio setting. In the second set of experiments (Section 5.3), where we investigate the performance of LCR relative to other recommendation systems, we report results with a fixed number of ratings (as in CofiRank). This choice was made to facilitate a comparison with perviously published results using the fixed number of ratings setting.

Evaluation Metrics.

We use three evaluation measures, which are widely used to evaluate ranking approaches. *Zero-one Error* is the average ratio of correctly ordered pairs of test items, averaged over all users and is based on the zero-one ranking loss (6) (giving constant loss when the relative order of preferences contradicts the ground truth):

$$\mathcal{E}_{0/1} = \sum_{u\in\mathcal{U}} \sum_{i\in T_u} \sum_{j\in M_u\cup T_u\setminus\{i\}} \frac{1}{Z_u} \mathbf{1}[\Delta M_{u,i,j}\cdot g(u,i,j) < 0],$$

where $Z_u = |U|\cdot|T_u|\cdot(|M_u\cup T_u|-1)$, and M_u and T_u are the set of available ratings for user u in the train and test sets respectively.

Note that we compare the relative order of an item in the test set with all other known ratings, including those in training set. In other words, pairs of items belonging to the train and test sets are used as well as pairs of items that belong only to the test set. Pairs of items only belonging to the train set are not used for testing.

Average Precision is defined as the area under the precision-recall curve, and is given by

$$AvgP = \frac{1}{|U|}\sum_{u\in U}\int_0^1 P(r)dr,$$

where the integration variable r ranges over all possible recall levels and $P(r)$ is the precision at recall r. In practice, the integral is replaced with a finite sum over every position in the ranked sequence of recommendations.

DCG@k takes into account the order of recommended items in the list by discounting the importance and is formally given by

$$DCG@k = \sum_{i=1}^{k} \frac{2^{rel_i}-1}{\log_2(i+1)},$$

where i ranges over positions in the recommendation list and rel_i indicates how much the i-item is relevant to the user (we use the observed rating score for this quantity). *NDCG* is the ratio of DCG to the maximum possible DCG for that user. This maximum occurs when the recommended items are presented in decreasing order of user preference. NDCG is perhaps the most popular evaluation metric, reflecting the importance of retrieving good items at the top of the ranking. In our experiments we use NDCG with $k = 10$.

Datasets.

We used three real-world datasets. MovieLens 100K[1] contains 943 users and 1,682 items with 6.3% (user, item) pairs rated (or 6.3% dense). This small dataset is ideal for quickly benchmarking several models, so we used it to explore the importance of various hyper-parameters. We also used the bigger EachMovie dataset, with 61,265 users, 1,648 items, and a density 2.8% ratings. These two datasets are widely used in the literature. In addition, we used the Yelp[2] dataset (43,873 users and 11,537 items, with 0.04% ratings), which was released for the Yelp business rating prediction challenge at ACM RecSys 2013. This dataset is the most recent one, reflecting a recent trend of extreme sparsity.

Model Parameters.

We varied the number of local models in the range {10, 20, 30, 40, 50}, the rank or latent space dimension in the set {5, 10, 15, 20}, and the loss functions $\{\mathcal{L}_{\text{Log[M]}}, \mathcal{L}_{\text{Log[A]}}, \mathcal{L}_{\text{Exp[M]}}, \mathcal{L}_{\text{Hinge[A]}}\}$ in Table 1 with $\gamma = 0$. We used Epanechnikov kernel as a smoothing kernel as it achieves the lowest integrated squared error [41] (with kernel bandwidth of 0.8). We also used L_2-regularization, where the regularization coefficient was determined by cross-validation. Cross validation was also used to determine early stopping. In the cross validation procedure, we set aside 20% of the training data for validation purpose. Note that different evaluation metrics typically lead to different stopping times. In our experiments, we stopped either when the improvement in training error got smaller than some threshold (0.001) or when the algorithm reached 200 iterations.

5.2 Trend Analysis

We first show how our LCR model behaves with different combinations of parameters. Specifically, we vary the rank of each local model in $\{5, 10, 15, 20\}$, and the number of local models in $\{10, 20, 30, 40, 50\}$. We used the MovieLens 100K dataset with fixed ratio split (ratio = 0.5) for this part of the experiments.

Effect of the Rank.

[1]http://www.grouplens.org/

[2]http://recsys.acm.org/recsys13/recsys-2013-challenge-workshop/

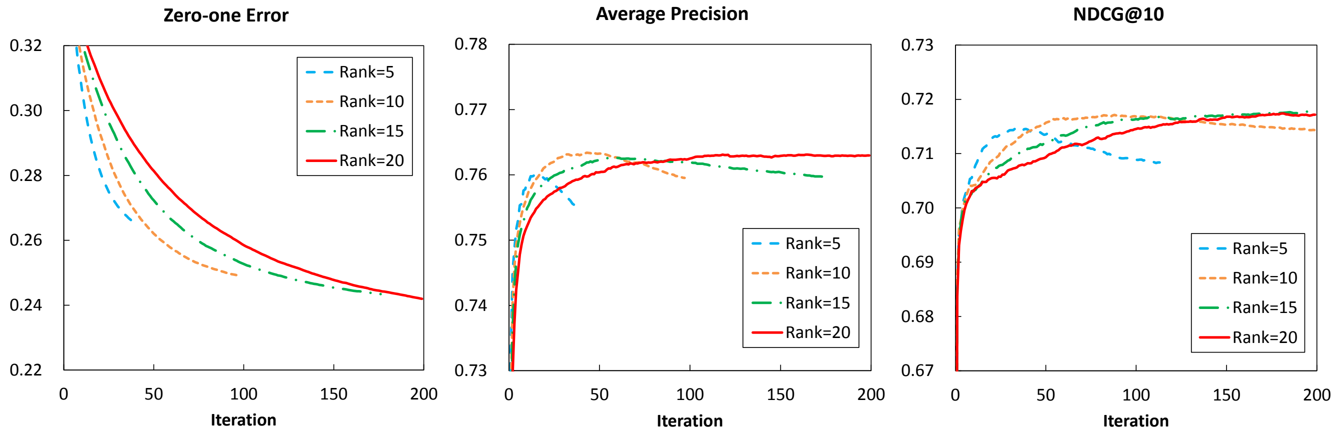

Figure 1: Effect of the model rank on the performance of LCR, measured by zero-one error (left), average precision (middle), and NDCG@10 (right). Ranks vary in $\{5, 10, 15, 20\}$, while the number of local models is fixed to 10. The optimized loss in the training procedure is $\mathcal{L}_{\text{Log[M]}}$ in Table 1 with $\gamma = 0$.

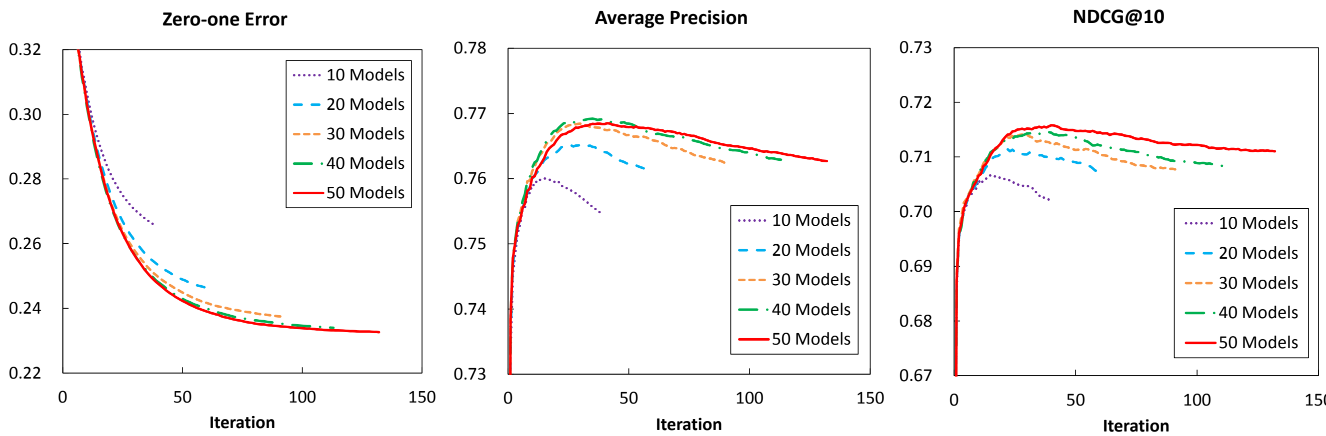

Figure 2: Effect of the number of local models on the performance of LCR, measured by zero-one error (left), average precision (middle), and NDCG@10 (right). The number of local models vary in $\{10, 20, 30, 40, 50\}$, while the rank of each local model is fixed to 5. The optimized loss in the training procedure is $\mathcal{L}_{\text{Log[M]}}$ in Table 1 with $\gamma = 0$.

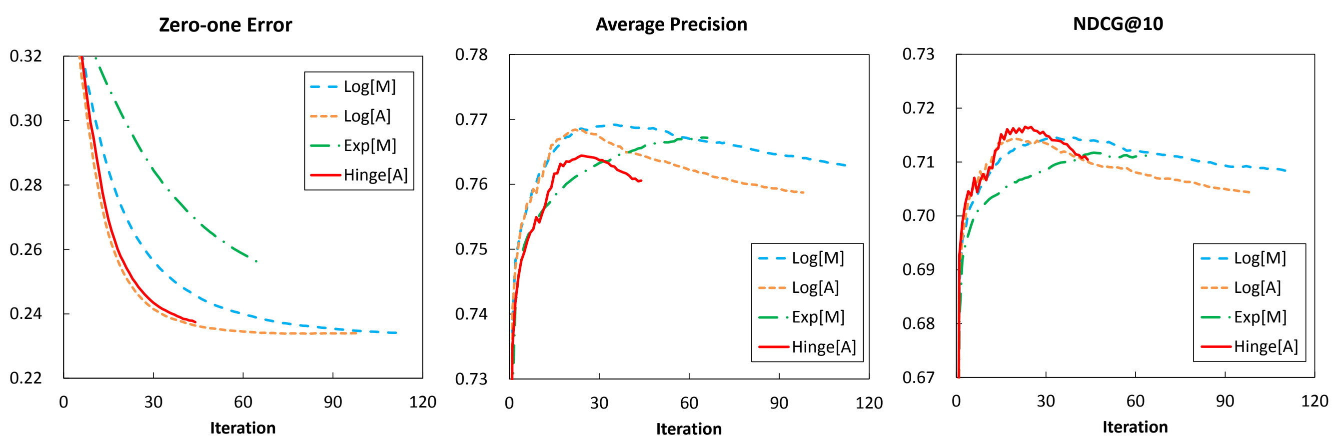

Figure 3: Effect of the optimized loss on the performance of LCR, measured in zero-one error (left), average precision (middle), and NDCG@10 (right). We tried $\{\mathcal{L}_{\text{Log[M]}}, \mathcal{L}_{\text{Log[A]}}, \mathcal{L}_{\text{Exp[M]}}, \mathcal{L}_{\text{Hinge[A]}}\}$ in Table 1 with $\gamma = 0$, while the rank of each local model and the number of local models is fixed to 5 and 40, respectively.

Figure 1 compares the performance of LCR when varying the rank of the local models. The zero-one error (left) decreases monotonically as expected, since we minimize a smooth surrogate $\mathcal{L}_{\text{Log[M]}}$ loss. When the rank gets higher, the final performance gets better, but it takes more iterations until convergence. In other words, higher ranks achieve better performance, but take longer to converge. Average precision and NDCG@10 show a different trend. Unlike the zero-one error, they overfit at some point, implying that it is important to use with these evaluation metrics cross validation for determining early stopping. With higher ranks, however, overfitting seems less of a problem; when the rank is sufficiently large, it achieves its best performance at the end. Note that similar results are obtained for other loss functions and number of local models.

Effect of the Number of Local Models.

Figure 2 compares the performance of LCR when varying the number of local models. As in Figure 1, the zero-one error monotonically decreases and converges to an improved score as the number of local models increases. Average precision and NDCG@10 tend to overfit, but the tendency to overfit diminishes as the number of local models increases. In contrast to Figure 1, however, more local models tend to result in almost always better performance than less local models. This shows that increasing the number of local models is relatively resistant to overfitting, unlike increasing the decomposition rank (compare the right two panels of Figure 1 and 2).

Effect of the Optimized Loss Function.

We also compared how LCR performs when changing the surrogate loss function that is used in the optimization procedure during the training stage. Figure 3 compares the performance of LCR optimized with different surrogate losses. Training on log-losses, in general, shows outstanding performances, as evaluated by zero-one error. Note that $\mathcal{L}_{\text{Log[A]}}$ achieves its best performance earlier than $\mathcal{L}_{\text{Log[M]}}$, but tends to overfit more. On the other hand, training by minimizing $\mathcal{L}_{\text{Exp[M]}}$ overfits much less as evaluated by both average precision and NDCG. Training by minimizing $\mathcal{L}_{\text{Hinge[A]}}$ was the quickest to converge among the four and performed especially well when evaluated in terms of the zero-one error. A drawback of $\mathcal{L}_{\text{Hinge[A]}}$ is its significant overfitting when evaluated by average precision and NDCG.

Figure 4 illustrates the behavior of the different loss functions. $\mathcal{L}_{\text{Log[M]}}$ loss (left-most) assigns almost no penalty when the estimation is correct, while relatively high penalty for wrong prediction. This means a model minimizing $\mathcal{L}_{\text{Log[M]}}$ loss focuses on pairs which are currently incorrectly estimated, while almost ignoring other pairs. $\mathcal{L}_{\text{Exp[M]}}$ loss (third) is an even more extreme loss function. On the other hand, $\mathcal{L}_{\text{Log[A]}}$ and $\mathcal{L}_{\text{Hinge[A]}}$ losses assign non-zero penalty for correctly ordered pairs if the degree does not match precisely.

We also tried $\mathcal{L}_{\text{Exp[A]}}$ and $\mathcal{L}_{\text{Hinge[M]}}$ in Table 1, but we excluded them for the following reasons. $\mathcal{L}_{\text{Exp[A]}}$ loss was too sensitive to hyper-parameters such as learning rate, making it difficult to use in practical settings. $\mathcal{L}_{\text{Exp[A]}}$ is more extreme than $\mathcal{L}_{\text{Exp[M]}}$, and thus is highly influenced by a very small portion of training examples. Models trained on $\mathcal{L}_{\text{Hinge[M]}}$ achieve performance similar to but slightly worse than models trained on $\mathcal{L}_{\text{Hinge[A]}}$.

Overall Comparison.

As a summary, Figure 5 compares the final performance when varying the rank and the number of local models. Note that (1) higher rank generally leads to better performance with any evaluation metric, and (2) more local models also leads to better performance with all evaluation metrics. Increasing the rank and number of local models, however, also increases computation time.

5.3 Comparison with Other Methods

Experimental Setup.

In this section, we compare the performance of LCR with other models. Specifically, we compare with standard matrix factorization models (Regularized SVD [2], NMF [24], Bayesian PMF [35]), with least squares local matrix approximation (LLORMA [25]), and with the ranked loss minimization methods CofiRank [43] and GCR (matrix approximation minimizing ranked loss in Section 4.2).

Among the different baselines above, LLORMA is notable since it is a local matrix approximation approach (though based on least squares minimization), and GCR is notable since it is a global matrix approximation based on ranked loss minimization. CofiRank is notable as it is considered a very strong baseline in recent literature. We implemented all of the methods above within the PREA toolkit [28], with the exception of CofiRank that made its code publicly available.

In order to compare with previous published results, we adopt here the CofiRank weak generalization setup (see Section 6 of [43]), predicting the rank of unrated items for users known at training time. For each user, N randomly chosen items, with $N = 10, 20, 50$, are used for training. Another 10 items are set aside for validation set, and all other items are used for testing. Users with less than $N + 20$ ratings are dropped to guarantee at least 10 items can be used for testing. This is a slight modification of the original CofiRank experimental setup, where no extra validation set was used. This modification was also proposed in [40, 12].

Parameter Setting.

For LCR, we consider all options including rank, number of local models, and optimized loss function as hyper-parameters. That is, we select the best model for each dataset and for each evaluation metric on the separate validation set. For CofiRank, we use the same parameter values (100 dimensions and $\lambda = 10$) provided in the original paper [43], and default values provided in the source code for unstated parameters (such as the maximum number of iterations and BMRM parameters). For LLORMA, the rank of local models was chosen among $\{5, 10, 15, 20\}$, and the number of local models among $\{10, 20, 30, 40, 50\}$. We used the Epanechnikov kernel with width 0.8. All other parameters were set to default values in their implementation. Ranks of other matrix factorization methods were set to $\{10, 20, 30, 40, 50\}$ (SVD), $\{20, 40, 60, 80, 100\}$ (NMF), and $\{2, 3, 4, 5\}$ (BPMF). Regularization coefficients and learning rates were chosen by cross validation.

Result and Discussion.

Table 2 compares LCR with competing approaches in terms of average precision and NDCG@10.

In terms of average precision, LCR outperforms all other approaches on MovieLens and Yelp datasets. With Each-

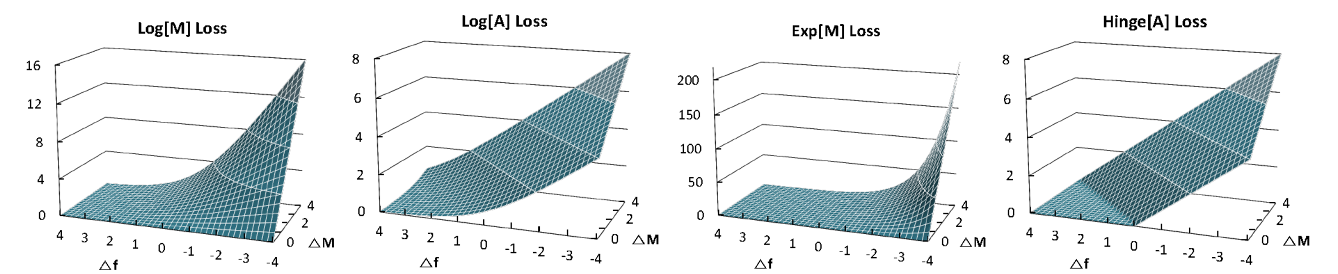

Figure 4: Shape of loss functions used in our experiments. ΔM is assumed to be always positive by arranging two items such that $M_{u,i} > M_{u,j}$. Δf may be either positive (left-half of each plot) or negative (right-half of each plot); positive Δf means the estimation is concordant with the real preference, while negative Δf implies the preference order is reversed in our estimation. Thus, as we expect, each loss function assigns higher penalty for more significant mistakes (in this figure, larger ΔM for negative Δf).

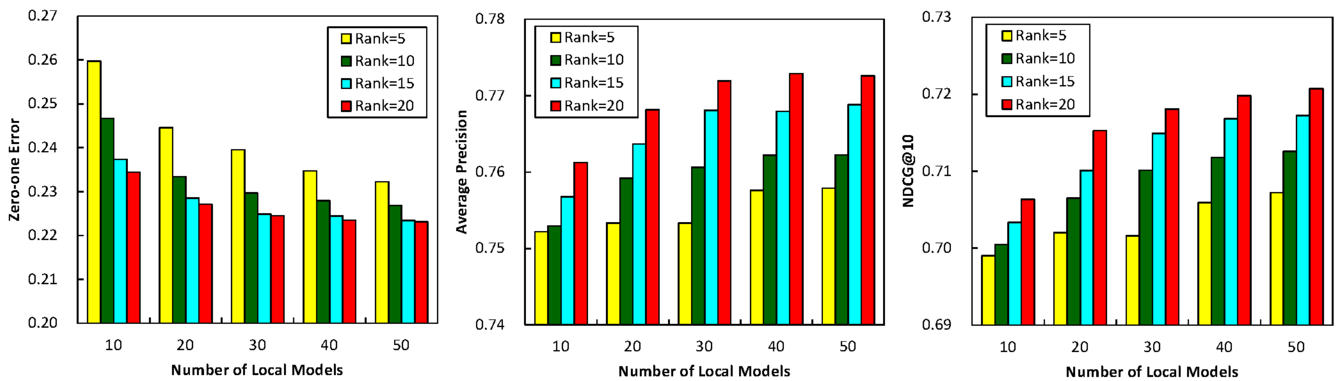

Figure 5: Performance trend in zero-one error (Left; the lower the better), average precision (Middle; the higher the better), and NDCG@10 (Right; the higher the better), with various ranks and number of local models. The presented results are with $\mathcal{L}_{\mathrm{Log[A]}}$. Models with other losses show similar trend.

Movie, it lags behind CofiRank in some cases ($N = 10, 20$), but is better than others. In terms of NDCG, LCR performs better for all datasets as well, with the exception of EachMovie with $N = 50$. We conclude that LCR generally outperforms other approaches in terms of ranking metrics, such as average precision and NDCG, which are motivated by the ranking scenario of modern recommendation systems.

The improvement of LCR is particularly notable for very sparse datasets such as the Yelp dataset, and when the amount of available training points (N) is small. The improvement of LCR over GCR suggests that the locality assumption is more plausible and that the observed matrix is not well approximated by a single low-rank matrix. The improvement of LCR over LLORMA indicates that in ranking scenario where evaluation is measured using average precision or NDCG, it makes more sense to use ranked loss minimization than least squares.

6. RELATED WORK

Recommendation systems emerged in 1990s in order to provide personalized services in E-commerce. They have been popularized by the very successful Netflix competition[3] held between 2006 and 2009, and since then many algorithms were proposed to for predicting ratings and constructing recommendations.

[3] http://www.netflixprize.com/

Collaborative filtering (CF) is one specific approach to recommendation systems that does not use any information about users or items beyond the users and items IDs. Early CF algorithms included memory based approaches that predict the rating of items based on the similarity between users [3, 18] or items [36] and model-based approaches that build a rating prediction model from the training data. Proposed model-based approaches include incomplete SVD [2], non-negative matrix factorization [24], Bayesian extensions [35, 34], and maximum margin versions [33]. Lee et al. [27] conducted a comparative study with a number of state-of-the-art and traditional recommendation algorithms using the PREA toolkit [28].

In the early 2000s, machine learning techniques were developed specifically for ranking problems (see for instance [19]). While early approaches focused on minimizing the pairwise zero-one loss, more recent approaches were developed to learn objectives that better follow metrics like NDCG or average precision [43, 8], which are more in line with real applications like product recommendation. Recent examples include [1, 40], which proposed collaborative ranking algorithms to approximately optimize NDCG directly; EigenRank [29], which optimizes a preference function using Kendall's Tau rank correlation. Additional examples are [16, 31, 42, 12, 37].

The idea of collaborative ranking was developed and much more widely-used for web search and label ranking. Mini-

Metric		Average Precision			NDCG@10		
Data \ Method		$N = 10$	$N = 20$	$N = 50$	$N = 10$	$N = 20$	$N = 50$
MovieLens	CofiRank	0.6632 ± 0.0020	0.6825 ± 0.0034	0.6915 ± 0.0021	0.6502 ± 0.0021	0.6629 ± 0.0040	0.6912 ± 0.0009
	RegSVD	0.7204 ± 0.0021	0.7321 ± 0.0047	0.7501 ± 0.0045	0.6425 ± 0.0073	0.6510 ± 0.0066	0.6778 ± 0.0072
	NMF	0.7107 ± 0.0017	0.7234 ± 0.0041	0.7328 ± 0.0052	0.6285 ± 0.0069	0.6336 ± 0.0097	0.6471 ± 0.0064
	BPMF	0.6376 ± 0.0115	0.6517 ± 0.6517	0.6915 ± 0.0064	0.5636 ± 0.0119	0.5513 ± 0.0184	0.5956 ± 0.0035
	LLORMA	0.7341 ± 0.0016	0.7424 ± 0.0015	0.7490 ± 0.0053	0.6712 ± 0.0027	0.6682 ± 0.0061	0.6746 ± 0.0063
	GCR	0.7209 ± 0.0011	0.7356 ± 0.0010	0.7534 ± 0.0021	0.6990 ± 0.0011	0.6908 ± 0.0050	0.6932 ± 0.0043
	LCR	$\mathbf{0.7406 \pm 0.0019}$	$\mathbf{0.7503 \pm 0.0022}$	$\mathbf{0.7626 \pm 0.0017}$	$\mathbf{0.7152 \pm 0.0056}$	$\mathbf{0.7022 \pm 0.0049}$	$\mathbf{0.7039 \pm 0.0038}$
EachMovie	CofiRank	$\mathbf{0.7491 \pm 0.0056}$	$\mathbf{0.7476 \pm 0.0054}$	0.7231 ± 0.0033	0.6635 ± 0.0058	0.6985 ± 0.0031	$\mathbf{0.7158 \pm 0.0038}$
	RegSVD	0.6979 ± 0.0011	0.7159 ± 0.0009	$\mathbf{0.7307 \pm 0.0020}$	0.6632 ± 0.0009	0.6784 ± 0.0015	0.6981 ± 0.0012
	NMF	0.6904 ± 0.0011	0.6895 ± 0.0002	0.6769 ± 0.0015	0.6555 ± 0.0014	0.6542 ± 0.0007	0.6385 ± 0.0022
	BPMF	0.6678 ± 0.0143	0.7024 ± 0.0054	0.7172 ± 0.0012	0.6261 ± 0.0170	0.6615 ± 0.0053	0.6742 ± 0.0007
	LLORMA	0.7151 ± 0.0040	0.7116 ± 0.0032	0.7258 ± 0.0020	0.6820 ± 0.0033	0.6778 ± 0.0035	0.6863 ± 0.0028
	GCR	0.7088 ± 0.0006	0.7106 ± 0.0010	0.6958 ± 0.0006	0.6998 ± 0.0008	0.6979 ± 0.0010	0.6779 ± 0.0012
	LCR	0.7307 ± 0.0010	0.7372 ± 0.0018	$\mathbf{0.7330 \pm 0.0045}$	$\mathbf{0.7166 \pm 0.0017}$	$\mathbf{0.7201 \pm 0.0010}$	0.6988 ± 0.0046
Yelp	CofiRank	0.7246 ± 0.0018	0.7273 ± 0.0020	0.7235 ± 0.0035	0.6997 ± 0.0026	0.6842 ± 0.0029	0.6680 ± 0.0028
	RegSVD	0.7799 ± 0.0013	0.7829 ± 0.0010	$\mathbf{0.7774 \pm 0.0034}$	0.7021 ± 0.0019	0.6953 ± 0.0021	0.6823 ± 0.0035
	NMF	0.7757 ± 0.0026	0.7799 ± 0.0012	0.7750 ± 0.0031	0.6926 ± 0.0032	0.6871 ± 0.0037	0.6705 ± 0.0051
	BPMF	0.7063 ± 0.0041	0.7035 ± 0.0020	0.7035 ± 0.0061	0.6191 ± 0.0022	0.5983 ± 0.0032	0.5838 ± 0.0029
	LLORMA	0.7844 ± 0.0011	$\mathbf{0.7882 \pm 0.0012}$	0.7822 ± 0.0024	0.7065 ± 0.0023	0.7019 ± 0.0023	0.6842 ± 0.0039
	GCR	0.7754 ± 0.0012	0.7797 ± 0.0034	0.7428 ± 0.0047	0.7465 ± 0.0023	0.7332 ± 0.0026	0.6582 ± 0.0072
	LCR	$\mathbf{0.7903 \pm 0.0021}$	$\mathbf{0.7901 \pm 0.0040}$	$\mathbf{0.7791 \pm 0.0043}$	$\mathbf{0.7575 \pm 0.0024}$	$\mathbf{0.7425 \pm 0.0018}$	$\mathbf{0.7212 \pm 0.0051}$

Table 2: Test performance in terms of average precision and NDCG@10. Higher values mean better performance. Bold faces mean that the method performs statistically significantly better in the setting, at the level of 95% confidence level.

mizing pair-wise ranking loss was proposed in Ranking SVM [17], RankBoost [13], RankNet [4], and FRank [39]. Wsabie [44] used weighted approximate-rank pairwise loss in label ranking for image annotation problem. Direct list-wise optimization over NDCG or average precision was also intensely investigated, with a few examples being LambdaRank [11], SVM$^{\text{RANK}}$ [48], AdaRank [46], and [5, 47]. Other examples of collaborative ranking in web search are [14, 7, 6, 30].

7. SUMMARY AND FUTURE WORK

We presented a novel collaborative ranking method based on the assumption that the rating matrix is globally high-rank but locally low-rank. The LCR approach generalizes LLORMA [25] from least squares to ranked loss minimization, and it outperforms LLORMA and state-of-the-art ranked loss minimization methods for collaborative filtering.

It was previously observed that the local low-rank assumption applies to recommendation systems in the context of least squares rating prediction. In this paper we verify that the same conclusion holds when performance is evaluated using a variety of ranked loss functions. Besides providing improved ranking accuracy, our model also scales up to large datasets due to our proposed parallel training scheme. Another computational advantage stems from the fact that each local model may be restricted to a rank that is lower than the rank selected for a global model. Indeed, replacing a single rank l matrix factorization with multiple rank l' (local) matrix factorizations with $l' < l$ may lead to a win-win situation in terms of both computational efficiency and ranking accuracy.

In the proposed LCR algorithm, we randomly chose the anchor points of the local models. Even though this practice works well in experiments, it is interesting to investigate ways of adaptively selecting anchor points or even jointly selecting anchor points and training the local models.

8. REFERENCES

[1] S. Balakrishnan and S. Chopra. Collaborative ranking. In *Proc. of the ACM International Conference on Web Search and Data Mining*, 2012.

[2] D. Billsus and M. J. Pazzani. Learning collaborative information filters. In *Proc. of the International Conference on Machine Learning*, 1998.

[3] J. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proc. of Uncertainty in Artificial Intelligence*, 1998.

[4] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *Proc. of the International Conference on Machine Learning*, 2005.

[5] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *Proc. of the International Conference on Machine Learning*, 2007.

[6] Z. Chen and H. Ji. Collaborative ranking: a case study on entity linking. In *Proc. of the Conference on Empirical Methods in Natural Language Processing*, 2011.

[7] B. Chidlovskii, N. S. Glance, and M. A. Grasso. Collaborative re-ranking of search results. In *Proc. of AAAI-2000 Workshop on AI for Web Search*, 2000.

[8] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proc. of the ACM Conference on Recommender Systems*, 2010.

[9] O. Dekel, C. Manning, and Y. Singer. Log-linear models for label ranking. *Advances in Neural Information Processing Systems*, 2003.

[10] J. Dillon, Y. Mao, G. Lebanon, and J. Zhang. Statistical translation, heat kernels, and expected distances. In *Uncertainty in Artificial Intelligence*, pages 93–100. AUAI Press, 2007.

[11] P. Donmez, K. M. Svore, and C. J. Burges. On the local optimality of lambdarank. In *Proc. of International ACM SIGIR Conference*, 2009.

[12] C. Fan, Y. Lan, J. Guo, Z. Lin, and X. Cheng. Collaborative factorization for recommender systems. In *Proc. of the International ACM SIGIR Conference*, 2013.

[13] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4:933–969, 2003.

[14] J. Freyne, B. Smyth, M. Coyle, E. Balfe, and P. Briggs. Further experiments on collaborative ranking in community-based web search. *Artificial Intelligence Review*, 21(3-4):229–252, 2004.

[15] S. Hacker and L. von Ahn. Matchin: eliciting user preferences with an online game. In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems*, 2009.

[16] E. Harrington. Online ranking/collaborative filtering using the perceptron algorithm. In *Proc. of the International Conference on Machine Learning*, 2003.

[17] R. Herbrich, T. Graepel, and K. Obermayer. Large margin rank boundaries for ordinal regression. In *Advances in Neural Information Processing Systems*, 1999.

[18] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *Proc. of ACM SIGIR Conference*, 1999.

[19] T. Joachims. Optimizing search engines using clickthrough data. In *Proc. of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002.

[20] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proc. of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008.

[21] N. D. Lawrence and R. Urtasun. Non-linear matrix factorization with gaussian processes. In *Proc. of the International Conference on Machine Learning*, 2009.

[22] G. Lebanon. Learning Riemannian metrics. In *Proc. of the 19th Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 2003.

[23] G. Lebanon. Axiomatic geometry of conditional models. *IEEE Transactions on Information Theory*, 51(4):1283–1294, 2005.

[24] D. Lee and H. Seung. Algorithms for non-negative matrix factorization. In *Advances in Neural Information Processing Systems*, 2001.

[25] J. Lee, S. Kim, G. Lebanon, and Y. Singer. Local low-rank matrix approximation. In *Proc. of the International Conference on Machine Learning*, 2013.

[26] J. Lee, M. Sun, S. Kim, and G. Lebanon. Automatic feature induction for stagewise collaborative filtering. In *Advances in Neural Information Processing Systems*, 2012.

[27] J. Lee, M. Sun, and G. Lebanon. A comparative study of collaborative filtering algorithms. *ArXiv Report 1205.3193*, 2012.

[28] J. Lee, M. Sun, and G. Lebanon. Prea: Personalized recommendation algorithms toolkit. *Journal of Machine Learning Research*, 13:2699–2703, 2012.

[29] N. N. Liu and Q. Yang. Eigenrank: a ranking-oriented approach to collaborative filtering. In *Proc. of the International ACM SIGIR Conference*, 2008.

[30] A. Mohan, Z. Chen, and K. Q. Weinberger. Web-search ranking with initialized gradient boosted regression trees. *Journal of Machine Learning Research-Proceedings Track*, 14:77–89, 2011.

[31] S. Park and D. M. Pennock. Applying collaborative filtering techniques to movie search for better ranking and browsing. In *Proc. of the ACM SIGKDD International Conference*, 2007.

[32] Y. Park and A. Tuzhilin. The long tail of recommender systems and how to leverage it. In *Proc. of the ACM Conference on Recommender Systems*, 2008.

[33] J. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *Proc. of the International Conference on Machine Learning*, 2005.

[34] R. Salakhutdinov and A. Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proc. of the International Conference on Machine Learning*, 2008.

[35] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems*, 2008.

[36] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *Proc. of the International Conference on World Wide Web*, 2001.

[37] 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 *Proc. of the ACM Conference on Recommender Systems*, 2012.

[38] T. F. Tan and S. Netessine. Is tom cruise threatened? using netflix prize data to examine the long tail of electronic commerce. *Wharton Business School, University of Pennsylvania, Philadelphia*, 2009.

[39] M.-F. Tsai, T.-Y. Liu, T. Qin, H.-H. Chen, and W.-Y. Ma. Frank: a ranking method with fidelity loss. In *Proc. of International ACM SIGIR Conference*, 2007.

[40] M. Volkovs and R. S. Zemel. Collaborative ranking with 17 parameters. In *Advances in Neural Information Processing Systems*, 2012.

[41] M. P. Wand and M. C. Jones. *Kernel Smoothing*. Chapman and Hall/CRC, 1995.

[42] J. Wang, S. Robertson, A. P. de Vries, and M. J. Reinders. Probabilistic relevance ranking for collaborative filtering. *Information Retrieval*, 11(6):477–497, 2008.

[43] M. Weimer, A. Karatzoglou, Q. V. Le, and A. Smola. Cofi rank: Maximum margin matrix factorization for collaborative ranking. In *Advances in Neural Information Processing Systems*, 2007.

[44] J. Weston, S. Bengio, and N. Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *Proc. of the International Joint Conference on Artificial Intelligence*, 2011.

[45] E. Xing, A. Ng, M. Jordan, and S. Russel. Distance metric learning with applications to clustering with side information. In *Advances in Neural Information Processing Systems*, 2003.

[46] J. Xu and H. Li. Adarank: a boosting algorithm for information retrieval. In *Proc. of International ACM SIGIR Conference*, 2007.

[47] J. Xu, T.-Y. Liu, M. Lu, H. Li, and W.-Y. Ma. Directly optimizing evaluation measures in learning to rank. In *Proc. of International ACM SIGIR Conference*, 2008.

[48] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *Proc. of International ACM SIGIR Conference*, 2007.