A Harmonic Extension Approach for Collaborative Ranking

Da Kuang* Zuoqiang Shi† Stanley Osher* Andrea Bertozzi*

Abstract

We present a new perspective on graph-based methods for collaborative ranking for recommender systems. Unlike user-based or item-based methods that compute a weighted average of ratings given by the nearest neighbors, or low-rank approximation methods using convex optimization and the nuclear norm, we formulate matrix completion as a series of semi-supervised learning problems, and propagate the known ratings to the missing ones on the user-user or item-item graph globally. The semi-supervised learning problems are expressed as Laplace-Beltrami equations on a manifold, or namely, *harmonic extension*, and can be discretized by a point integral method. We show that our approach does not impose a low-rank Euclidean subspace on the data points, but instead minimizes the dimension of the underlying manifold. Our method, named *LDM (low dimensional manifold)*, turns out to be particularly effective in generating rankings of items, showing decent computational efficiency and robust ranking quality compared to state-of-the-art methods.

1 Introduction

Recommender systems are crucial components in contemporary e-commerce platforms (Amazon, eBay, Netflix, etc.), and were popularized by the Netflix challenge. Detailed surveys of this field can be found in [11, 13]. Recommendation algorithms are commonly based on collaborative filtering, or "crowd of wisdom", and can be categorized into memory-based and model-based approaches. Memory-based approaches include user-based and item-based recommendation [18]. For example, for a user u, we retrieve the highly-rated items from the nearest neighbors of u, and recommend those items that have not been consumed by u. Memory-based methods are actually based on a graph, where a user-user or item-item similarity matrix defines the nearest neighbors of each user or item. In contrast, model-based methods are formulated as matrix completion problems which assume that the entire user-by-item rating matrix is low-rank [3], and the goal is to predict the missing ratings given the observed ratings. While memory-based methods are typically more computationally efficient, model-based methods can achieve much higher quality for collaborative filtering.

Popular model-based methods such as regularized SVD [3] minimize the sum-of-squares error over all the observed ratings. When evaluating the predictive accuracy of these algorithms, we often divide the whole data set into a training set and a test set. After obtaining a model on the training set, we evaluate the accuracy of the model's prediction on the test set in order to see how well it generalizes to unseen data. However, the measure for evaluating success in a practical recommender system is very different. What we care more about is whether the top recommended

*Department of Mathematics, University of California, Los Angeles (UCLA), Los Angeles, CA, 90095. Email: {dakuang,sjo,bertozzi}@math.ucla.edu.

†Yau Mathematical Sciences Center, Tsinghua University, Beijing, China, 100084. Email: zqshi@math.tsinghua.edu.cn.

items for a user u will actually be "liked" by u. In an experimental setting, the evaluation measure for success in this context is the resemblance between the ranked list of top recommended items and the ranked list of observed ratings in the test set. Thus, this measure that compares two rankings is more relevant to the performance in real scenarios. The problem that places priority on the top recommended items rather than the absolute accuracy of predicted ratings is referred to as top-N recommendation [5], or more recently *collaborative ranking* [9, 17], and is our focus in this paper.

We start with the matrix completion problem and formulate it as a series of semi-supervised learning problems, or in particular, harmonic extension problems on a manifold that can be solved by label propagation [24, 20]. For each item, we want to know the ratings by all the users, and the goal of the semi-supervised learning problem is to propagate the known labels for this item (observed ratings) to the unknown labels on the user-user graph; and reversely, for each user, to propagate the known labels given by this user to the unknown labels on the item-item graph.

Without loss of generality, we assume that there exists a user manifold, denoted as $\mathcal{M}$, which consists of an infinite number of users. In a user-by-item rating system with n items, each user is identified by an n-dimensional vector that consists of the ratings to n items. Thus, the user manifold $\mathcal{M}$ is a submanifold embedded in $\mathbb{R}^n$. For the i-th item, we define the rating function $f_i : \mathcal{M} \to \mathbb{R}$ that maps a user into the rating of this item.

One basic observation is that for a fixed item, *similar users should give similar ratings*. This implies that the function f_i, $1 \leq i \leq n$ is a *smooth* function on $\mathcal{M}$. Therefore, it is natural to find the rating function f_i by minimizing the following energy functional:

$$E(f) = \int_{\mathcal{M}} \|\nabla_{\mathcal{M}} f(u)\|^2 du, \tag{1}$$

where $\nabla_{\mathcal{M}} f(u)$ is the gradient at u defined on the manifold $\mathcal{M}$. Using standard variational approaches, minimizing the above functional (1) is reduced to solving the Laplace-Beltrami equation on the user manifold $\mathcal{M}$. Then the Laplace-Beltrami equation can be solved by a novel point integral method [20].

For the harmonic extension model, we also have an interpretation based on the low dimensionality of the user manifold, after which we call our method *LDM*. The user manifold is a manifold embedded in $\mathbb{R}^n$, and n is usually a large number. Compared to n, the intrinsic dimension of the user manifold is typically much smaller. Based on this observation, we use the dimension of the user manifold as a regularization to recover the rating matrix. This idea implies the following optimization problem:

$$\begin{aligned} \min_{\substack{X \in \mathbb{R}^{m\times n},\\ \mathcal{M} \subset \mathbb{R}^n}} \quad & \dim(\mathcal{M}), \\ \text{subject to:} \quad & P_\Omega(X) = P_\Omega(A), \quad \mathcal{U}(X) \subset \mathcal{M}, \end{aligned} \tag{2}$$

where $\dim(\mathcal{M})$ is the dimension of the manifold $\mathcal{M}$, and $\mathcal{U}(X)$ is the user set corresponding to the rows of X. $\Omega = \{(i, j) : \text{user } i \text{ rated item } j\}$ is the index set of the observed ratings, and P_Ω is the projection operator to Ω,

$$P_\Omega(X) = \begin{cases} x_{ij}, & (i,j) \in \Omega, \\ 0, & (i,j) \notin \Omega. \end{cases}$$

By referring to the theory in differential geometry, this optimization problem is reduced to the same formulation as that in harmonic extension (1) which gives our model a geometric interpretation.

Another important aspect of our proposed method is the weight matrix that defines the user-user or item-item graph. Because the information given in the rating matrix is incomplete, we can only assume that the weight matrix used in harmonic extension is a good guess. We will propose an efficient way to construct the weight matrix based on incomplete data.

Our main contribution is summarized as follows:

- We propose an algorithm that exploits manifold structures to solve the harmonic extension problem for collaborative ranking, representing a new perspective on graph-based methods for recommender systems.
- On real data sets, our method achieves robust ranking quality with reasonable run-time, compared to state-of-the-art methods for large-scale recommender systems.

The rest of this paper is organized as follows. In Section 2, we formulate the matrix completion problem as harmonic extension. In Section 3, we describe the point integral method to rigorously solve the discretized harmonic extension problem. In Section 4, we show that our approach seeks to minimize the dimension of the underlying manifold. In Section 5, we describe our more efficient way to compute the similarity matrix. In Section 6, we empirically demonstrate the efficiency and ranking quality of our method. In Section 7, we explain the connection and difference between our method and previous work. In Section 8, we discuss our proposed method and its implication on other methods.

Here are some notations we will use. For a vector $x = [x_1, \cdots, x_m]^T$, we call $y = [x_{i_1}, x_{i_2}, \cdots, x_{i_r}]^T$ a *subvector* of length r by extracting the elements of x in the index set $\{i_1, \cdots, i_r\}$, where $i_1 < i_2 < \cdots < i_r$. For a matrix M, a vector x, integers i, j, and sets of row and column indices S, S', we use $M_{i,j}, M_{S,S'}, M_{:,j}, M_{S,j}, x_S$ to denote an entry of M, a submatrix of M, the j-th column of M, a subvector of the j-th column of M, and a subvector of x, respectively.

2 Harmonic Extension Formulation

Consider a user-by-item rating matrix $A = (a_{ij}) \in \mathbb{R}^{m \times n}$, where rows correspond to m users, and columns correspond to n items. The observed ratings are indexed by the set $\Omega = \{(i, j) :$ user i rated item $j\}$. Let $\Omega_i = \{1 \leq j \leq m : (i, j) \in \Omega\}$, $1 \leq i \leq n$. Suppose there exists a "true" rating matrix A^* given by an oracle with no missing entries, which is not known to us, and $A|_\Omega = A^*|_\Omega$.

As mentioned in the introduction, we formulate matrix completion as a harmonic extension problem on a manifold. Recall the user manifold in Section 1, denoted as $\mathcal{M}$, which is embedded in $\mathbb{R}^n$. The set of m users in our user-by-item rating system is represented as $U = \{u_j, 1 \leq j \leq m\}$ where u_j is the j-th row of A^* and $U \subset \mathcal{M}$ is a sample of $\mathcal{M}$. Let $U_i = \{u_j \in U : j \in \Omega_i\}$ be the collection of users who rate the i-th item.

Then we compute the rating function f_i for all the users by minimizing the energy functional in (1):

$$\min_{f_i \in H^1(\mathcal{M})} \quad E(f_i) \quad \text{subject to:} \quad f_i(u_j)|_{U_i} = a_{ij}, \tag{3}$$

where H^1 is the Sobolev space. Hence, we need to solve the following type of optimization problem for n times.

$$\min_{f \in H^1(\mathcal{M})} \quad E(f) \quad \text{subject to:} \quad f(u)|_\Lambda = g(u), \tag{4}$$

where $\Lambda \subset \mathcal{M}$ is a point set.

To solve the above optimization problem, we first use the Bregman iteration to enforce the constraint [15].

Algorithm 1 Algorithm for solving (3)

```
repeat
    Get estimate for $\Delta_{\mathcal{M}}$ based on $f_1, f_2, \cdots, f_n$
    for $i = 1$ to $n$ do
        Solve (6) to obtain $f_i$
    end for
until some stopping criterion is satisfied
```

- Solve
$$f^{k+1} = \arg\min_f \; E(f) + \mu \|f - g + d^k\|^2_{L^2(\Lambda)}, \tag{5}$$
where $\|f\|^2_{L^2(\Lambda)} = \sum_{u \in \Lambda} |f(u)|^2$, d^n is a function defined on Λ.

- Update d^k,
$$d^{k+1}(u) = d^k(u) + f^{k+1}(u) - g(u), \; \forall u \in \Lambda.$$

- Repeat above process until convergence.

Using a standard variational approach, the solution to (5) can be reduced to the following Laplace-Beltrami equation:

$$\begin{cases} \Delta_{\mathcal{M}} f(\mathbf{x}) - \mu \sum_{\mathbf{y} \in \Lambda} \delta(\mathbf{x} - \mathbf{y})(f(\mathbf{y}) - h(\mathbf{y})) = 0, \; \mathbf{x} \in \mathcal{M}, \\ \dfrac{\partial f}{\partial \mathbf{n}}(\mathbf{x}) = 0, \quad \mathbf{x} \in \partial \mathcal{M}, \end{cases} \tag{6}$$

where δ is the Dirac-δ function in $\mathcal{M}$, $h = g - d^n$ is a given function on Λ, and $\mathbf{n}$ is the outer normal vector. That is to say, the function f that minimizes (4) is a harmonic function on $\mathcal{M} \backslash \partial \mathcal{M}$, and (6) is called the *harmonic extension* problem in the continuous setting.

If the underlying manifold $\mathcal{M}$ (the true rating matrix A^* in the discrete setting) were known, the n problems in (3) would be independent with each other and could be solved individually by (6). However, $\mathcal{M}$ is not known, and therefore we have to get a good estimate for the operator $\Delta_{\mathcal{M}}$ based on f_j's. Our algorithm for solving (3) is described on a high level in Algorithm 1, where we iteratively update f_j's and our estimate for $\Delta_{\mathcal{M}}$.

In the next section, we use the point integral method (PIM) to solve the Laplace-Beltrami equation (6).

Remark. The update of f_j's in Algorithm 1 follows the "Jacobi" scheme. We could also use the "Gauss-Seidel" scheme, i.e. re-estimate $\Delta_{\mathcal{M}}$ after the update of each f_j, but that would be much slower.

3 Point Integral Method (PIM)

Integral Equation: The key observation in PIM is that the Laplace-Beltrami operator has the following integral approximation:

$$
\begin{aligned}
&\int_{\mathcal{M}} w_t(\mathbf{x},\mathbf{y})\Delta_{\mathcal{M}} f(\mathbf{y})\mathrm{d}\mathbf{y} \\
\approx &-\frac{1}{t}\int_{\mathcal{M}} (f(\mathbf{x})-f(\mathbf{y}))w_t(\mathbf{x},\mathbf{y})\mathrm{d}\mathbf{y} \\
&+2\int_{\partial\mathcal{M}} \frac{\partial f(\mathbf{y})}{\partial \mathbf{n}} w_t(\mathbf{x},\mathbf{y})\mathrm{d}\tau_{\mathbf{y}},
\end{aligned} \tag{7}
$$

where $w_t(\mathbf{x},\mathbf{y}) = \exp(-\frac{|\mathbf{x}-\mathbf{y}|^2}{4t})$. The following theorem gives the accuracy of the integral approximation.

Theorem 3.1. *[19] If $f \in C^3(\mathcal{M})$ is a smooth function on $\mathcal{M}$, then for any $\mathbf{x} \in \mathcal{M}$,*

$$
\|r(f)\|_{L^2(\mathcal{M})} = O(t^{1/4}), \tag{8}
$$

where

$$
\begin{aligned}
r(f) = &\int_{\mathcal{M}} w_t(\mathbf{x},\mathbf{y})\Delta_{\mathcal{M}} f(\mathbf{y})\mathrm{d}\mathbf{y} \\
&+\frac{1}{t}\int_{\mathcal{M}} (f(\mathbf{x})-f(\mathbf{y}))w_t(\mathbf{x},\mathbf{y})\mathrm{d}\mathbf{y} \\
&-2\int_{\partial\mathcal{M}} \frac{\partial f(\mathbf{y})}{\partial \mathbf{n}} w_t(\mathbf{x},\mathbf{y})\mathrm{d}\tau_{\mathbf{y}}.
\end{aligned}
$$

Applying the integral approximation (7) to the Laplace-Beltrami equation (6), we get an integral equation

$$
\begin{aligned}
&\frac{1}{t}\int_{\mathcal{M}} (f(\mathbf{x})-f(\mathbf{y}))w_t(\mathbf{x},\mathbf{y})\mathrm{d}\mathbf{y} \\
&\qquad +\mu\sum_{\mathbf{y}\in\Lambda} w_t(\mathbf{x},\mathbf{y})(f(\mathbf{y})-h(\mathbf{y})) = 0,
\end{aligned} \tag{9}
$$

In this integral equation, there are no derivatives, and therefore it is easy to discretize over the point cloud.

Discretization: We notice that the closed form of the user manifold $\mathcal{M}$ is not known, and we only have a sample of $\mathcal{M}$, i.e. U. Next, we discretize the integral equation (7) over the point set U.

Assume that the point set U is uniformly distributed over $\mathcal{M}$. The integral equation can be discretized easily, as follows:

$$
\begin{aligned}
&\frac{|\mathcal{M}|}{m}\sum_{j=1}^{m} w_t(\mathbf{x}_i,\mathbf{x}_j)(f(\mathbf{x}_i)-f(\mathbf{x}_j))+ \\
&\qquad \mu t\sum_{\mathbf{y}\in\Lambda} w_t(\mathbf{x}_i,\mathbf{y})(f(\mathbf{y})-h(\mathbf{y})) = 0
\end{aligned} \tag{10}
$$

```
Algorithm 2 Harmonic Extension
Input: Initial rating matrix $A$.
Output: Rating matrix $R$.
Set $R = A$.
repeat
    Estimate the weight matrix $\boldsymbol{W} = (w_{ij})$ from the user set $U$ (Algorithm 3).
    Compute the graph Laplacian matrix: $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W}$
    for $i = 1$ to $n$ do
        repeat
            Solve the following linear systems
                $\boldsymbol{L}\boldsymbol{f}_i + \bar{\mu}\boldsymbol{W}_{:,U_i}(\boldsymbol{f}_i)_{U_i} = \bar{\mu}\boldsymbol{W}_{:,U_i}\boldsymbol{h}_{U_i},$
            where $\boldsymbol{h} = \boldsymbol{g}_i - \boldsymbol{d}^k$.
            Update $\boldsymbol{d}^k$,
                $\boldsymbol{d}^{k+1} = \boldsymbol{d}^k + \boldsymbol{f}^{k+1} - \boldsymbol{g}_i$
        until some stopping criterion for the Bregman iterations is satisfied
    end for
    $r_{ij} = f_i(u_j)$ and $R = (r_{ij})$.
until some stopping criterion is satisfied
```

where $|\mathcal{M}|$ is the volume of the manifold $\mathcal{M}$.

We can rewrite (10) in the matrix form.

$$\boldsymbol{L}\boldsymbol{f} + \bar{\mu}\boldsymbol{W}_{:,\Lambda}\boldsymbol{f}_\Lambda = \bar{\mu}\boldsymbol{W}_{:,\Lambda}\boldsymbol{h}. \tag{11}$$

where $\boldsymbol{h} = (h_1, \cdots, h_m)$ and $\bar{\mu} = \frac{\mu t m}{|\mathcal{M}|}$. $\boldsymbol{L}$ is a $m \times m$ matrix which is given as

$$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W} \tag{12}$$

where $\boldsymbol{W} = (w_{ij}),\ i, j = 1, \cdots, m$ is the weight matrix and $\boldsymbol{D} = \text{diag}(d_i)$ with $d_i = \sum_{j=1}^{m} w_{ij}$.

Remark. In the harmonic extension approach, we use a continuous formulation based on the underlying user manifold. And the point integral method is used to solve the Laplace-Beltrami equation on the manifold. If a graph model were used at the beginning, the natural choice for harmonic extension would be the graph Laplacian. However, it has been observed that the graph Laplacian is not consisitent in solving the harmonic extension problem [20, 16], and PIM gives much better results.

Remark. The optimization problem we defined in (1) can be viewed as a continuous analog of the discrete harmonic extension problem [24], which we write in our notations:

$$\min_{\boldsymbol{f}_i} \sum_{j,j'=1}^{n} w_{jj'} \left((\boldsymbol{f}_i)_j - (\boldsymbol{f}_i)_{j'}\right)^2 \quad \text{subject to:} \quad (\boldsymbol{f}_i)_{U_i} = A_{U_i,i}. \tag{13}$$

The formulation (13) in the context of collaborative ranking can be seen as minimizing the weighted sum of squared error in pairwise ranking. This form of loss function considers all the possible pairwise rankings of items, which is different from the loss function in previous work on collaborative ranking [9, 17]:

$$\sum_{j,j' \in U_i} \mathcal{L}\left([a_{ji} - a_{j'i}] - [(\boldsymbol{f}_i)_j - (\boldsymbol{f}_i)_{j'}]\right), \tag{14}$$

where $\mathcal{L}$ is a loss function such as hinge loss and exponential loss. Only the pairwise rankings of items in the *training set* are considered in (14).

4 Low Dimensional Manifold (LDM) Interpretation

In this section, we emphasize the other interpretation of our method based on the low dimensionality of the user manifold. In the user-by-item rating system, a user is represented by an n-dimensional vector that consists of the ratings to n items, and the user manifold is a manifold embedded in $\mathbb{R}^n$. Usually, n, the number of items, is a large number in the order of $10^3 \sim 10^6$. The intrinsic dimension of the user manifold is much less than n. Based on this observation, it is natural to recover the rating matrix by looking for the user manifold with the lowest dimension, which implies the optimization problem in (2):

$$\begin{aligned} \min_{\substack{X\in\mathbb{R}^{m\times n},\\ \mathcal{M}\subset\mathbb{R}^n}} \quad & \dim(\mathcal{M}), \\ \text{subject to:} \quad & P_\Omega(X) = P_\Omega(A), \quad \mathcal{U}(X) \subset \mathcal{M}. \end{aligned}$$

where $\dim(\mathcal{M})$ is the dimension of the manifold $\mathcal{M}$, and $\mathcal{U}(X)$ is the user set corresponding to the rows of X.

Next, we need to give a mathematical expression to compute $\dim(\mathcal{M})$. Here we assume $\mathcal{M}$ is a smooth manifold embedded in $\mathbb{R}^n$. Let $\alpha_i,\ i = 1, \cdots, d$ be the coordinate functions on $\mathcal{M}$, i.e.

$$\alpha_i(\mathbf{x}) = x_i, \quad \forall \mathbf{x} = (x_1, \cdots, x_n) \in \mathcal{M} \tag{15}$$

Using differential geometry, we have the following formula [16].

Proposition 4.1. *Let $\mathcal{M}$ be a smooth submanifold isometrically embedded in $\mathbb{R}^n$. For any $\mathbf{x} \in \mathcal{M}$,*

$$\dim(\mathcal{M}) = \sum_{i=1}^{n} \|\nabla_{\mathcal{M}} \alpha_i(\mathbf{x})\|^2$$

where $\nabla_{\mathcal{M}}$ is the gradient in the manifold $\mathcal{M}$.

We can clearly see that α_i corresponds to the rating function f_i. Using the above proposition, the manifold dimension minimization problem (2) can be rewritten as

$$\begin{aligned} \min_{\substack{X\in\mathbb{R}^{m\times n},\\ \mathcal{M}\subset\mathbb{R}^d}} \quad & \sum_{i=1}^{d} \|\nabla_{\mathcal{M}} f_i\|^2_{L^2(\mathcal{M})}, \\ \text{subject to:} \quad & f_i(x_j)|_{U_i} = a_{ij}, \quad \mathcal{U}(X) \subset \mathcal{M}, \end{aligned} \tag{16}$$

where

$$\|\nabla_{\mathcal{M}} f_i\|_{L^2(\mathcal{M})} = \left(\int_{\mathcal{M}} \|\nabla_{\mathcal{M}} f_i(\mathbf{x})\|^2 \mathrm{d}\mathbf{x} \right)^{1/2}. \tag{17}$$

This is the same optimization problem we solved in Section 2 and Section 3.

5 Weight Matrix

The weight matrix $\boldsymbol{W}$ plays an important role in our algorithm as well as other graph-based approaches [24, 18, 10, 7]. We employ the typical user-user or item-item graph with cosine similarity used in existing memory-based approaches for recommendation [18]. However, we have made substantial changes to make the procedure efficient for large sparse rating matrices. Our algorithm for building the weight matrix is described in detail in Algorithm 3. Again, we consider the user-user graph without loss of generality.

First, as usual, we can only afford to compute and store a sparse nearest-neighbor weight matrix. To get the K nearest neighbors for a target user u, traditional algorithms in memory-based methods require computing the distances between u and every other user, and selecting the K closest ones, where most of the computation is wasted if $K << m$. In our algorithm, we first identify the nearest neighbors approximately, without computing the actual distances or similarities, and then compute the similarities between u and its nearest neighbors only. We use a binary rating matrix R_B that records "rated or not-rated" information (Algorithm 3, line 1-2), and determine the K nearest neighbors using an *kd*-tree based approximate nearest neighbor algorithm (line 3, line 5) [12]. That is to say, two users who have rated similar sets of movies are more likely to be considered to be in each other's neighborhood, regardless of their numeric ratings for those movies. Neither of the ways to build the *kd*-tree and to find nearest neighbors based on the tree are as precise as a naïve search; however, empirical results in the next section have shown that our approximate strategy does not compromise the quality.

Second, we extended the VLFeat package [21] to enable building a *kd*-tree from a sparse data matrix (in our case, R_B) and querying the tree with sparse vectors. *kd*-tree uses a space partitioning scheme for efficient neighbor search [2]. For high-dimensional data, we employ the greedy way that chooses the most varying dimension for space partitioning at each step of building the tree [12], and the procedure terminates when each leaf partition has one data point. Thus, the complexity of building the tree is not exponential, contrary to common understanding; and the practical performance of *kd*-tree can be very efficient and better than that of locality sensitive hashing [1, 14]. For example, in our case with m data points in n dimensions, the complexity of building the tree is $O(m)$, rather than $O(2^n)$. In addition, when querying the tree, we put an upper bound D on the maximum number of distance comparisons, and therefore the overall complexity of finding K nearest neighbors for all the m data points is $O(Dm \log K)$ (the $\log K$ factor comes from maintaining a heap data structure for the K nearest neighbors).

Note that the resulting graph is not symmetric. Also the cosine similarity for two data points with incomplete information is defined by using the co-rated items only (Algorithm 3, line 8-9) [18].

6 Experiments

In this section, we evaluate our proposed method LDM in terms of both run-time and ranking quality. Since the volume of literature on recommender systems, collaborative filtering, and matrix completion is huge, we select only a few existing methods to compare with. All the experiments are run on a Linux laptop with one Intel i7-5600U CPU (4 physical threads) and 8 GB memory.

Algorithm 3 Building weight matrix from incomplete rating data

```
Input: Incomplete rating matrix $R \in \mathbb{R}^{m \times n}$, number of nearest neighbors $K$
Generate binary rating matrix $R_B \in \mathbb{R}^{m \times n}$:
    $(R_B)_{j,j'} = \begin{cases} 1, & R_{j,j'} \text{ is not missing} \\ 0, & R_{j,j'} \text{ is missing} \end{cases}$
Normalize each row of $R_B$ such that $\|(R_B)_{j,:}\|_2 = 1, \forall j, 1 \le j \le m$
Build a kd-tree on the data points (rows) in $R_B$
Initialize a sparse matrix $\boldsymbol{W} \leftarrow \mathbf{0}^{m \times m}$
for $j = 1$ to $m$ do
    $\mathcal{N}_B \leftarrow$ The set of $K$ approximate nearest neighbors of $(R_B)_{j,:}$, found by querying the kd-tree
    for $j' \in \mathcal{N}_B$ $(j' \neq j)$ do
        Set of co-rated items $\mathcal{C} \leftarrow \{i : R_{j,i}$ is not missing, and $R_{j',i}$ is not missing$\}$
        $\boldsymbol{W}_{j,j'} \leftarrow \text{cosine}(R_{j,\mathcal{C}}, R_{j',\mathcal{C}})$
    end for
end for
Output: Sparse weight matrix $\boldsymbol{W} \in \mathbb{R}^{m \times m}$
```

Table 1: Statistics of the data sets in our experiments. N is the number of ratings in the training set for each user. The total number of ratings in the training set is $N \times (\#$ users).

Data set		# users	# items	# ratings
MovieLens-100k	$N = 10$	943	1,682	100,000
	$N = 20$	744	1,682	95,269
MovieLens-1m	$N = 10$	6,040	3,706	1,000,209
	$N = 20$	5,289	3,701	982,040
	$N = 50$	3,938	3,677	924,053
MovieLens-10m	$N = 10$	69,878	10,677	10,000,054
	$N = 20$	57,534	10,675	9,704,223
	$N = 50$	38,604	10,672	8,897,688
	$N = 100$	24,328	10,666	7,730,011

6.1 Data Sets

We use three MovieLens[1] data sets in our experiments: *MovieLens-100k*, *MovieLens-1m*, and *MovieLens-10m*. In each of the data sets, each user has at least 20 ratings. Following the convention in previous work on collaborative ranking, we randomly select N ratings for each user as the training set, and the other ratings were used for testing. To keep at least 10 ratings in the testing set for each user, we remove the users with fewer than $N + 10$ ratings. After this preprocessing, we can generate several versions of these data sets with different N's, whose information is summarized in Table 1.

[1] http://grouplens.org/datasets/movielens/

6.2 Methods for Comparison

We compare our LDM method with singular value decomposition (SVD) as a baseline, and two state-of-the-art methods that are designed specifically for collaborative ranking. All the three methods for comparison optimize a pairwise ranking loss function. We do not perform hyperparameter selection on a separate validation set because it is time-consuming, but we investigate the effect of the hyperparameters in Section 6.4; and in Section 6.5, we use a fixed set of hyperparameters that can achieve a good balance between run-time and ranking quality, which are found empirically across several data sets. We list these methods below (their program options that will be used in Section 6.5 is in the footnote):

- *SVD*: We use the Java implementation of ranking-based SVD in the PREA toolkit[2,3]. This version uses gradient descent to optimize the ranking-based loss function (14), as opposed to the squared loss function in regularized SVD.
- *LCR*: Local collaborative ranking [9], as implemented in the PREA toolkit[4].
- *AltSVM*: Alternating support vector machine [17], as implemented in the `collranking` package[5]. We employ the default configurations.

Lastly, our proposed method LDM (Algorithm 2) is implemented in Matlab, and the construction and querying of the *kd*-tree (Algorithm 3) is implemented in C, for which we extended the VLFeat[6] package to build a *kd*-tree with a sparse input matrix efficiently. In Algorithm 2, we set $\mu = 1$ and run one inner iteration and one outer iteration only, since we empirically found that the weight matrix constructed from the incomplete input ratings by Algorithm 3 is often good enough for the algorithm to converge in one iteration. Algorithm 2 typically accounts for most ($\sim 95\%$) of the run-time in our method.

All the programs except SVD use 4 threads in our experiments, and are applied to the same training and testing random splits.

6.3 Evaluation Measure

We evaluate the ranking quality by *normalized discounted cumulative gain* (NDCG) @K [6], averaged over all the users. Given a ranked list of t items $i_1, \cdots, i_t$ and their *ground-truth* relevance score $r_{i_1}, \cdots, r_{i_t}$, the DCG@K score ($K \leq t$) is computed as

$$\text{DCG@}K(i_1, \cdots, i_t) = \sum_{j=1}^{K} \frac{2^{r_{i_j}} - 1}{\log_2(j+1)}. \tag{18}$$

Then, we sort the list of items in the decreasing order of the relevance score and obtain the list $i_1^*, \cdots, i_t^*$, where $r_{i_1^*} \geq \cdots \geq r_{i_t^*}$, and this sorted list achieves the maximum DCG@K score over all the possible permutations. The NDCG score is defined as the normalized version of (18):

$$\text{NDCG@}K(i_1, \cdots, i_t) = \frac{\text{DCG@}K(i_1, \cdots, i_t)}{\text{DCG@}K(i_1^*, \cdots, i_t^*)}. \tag{19}$$

[2] `http://prea.gatech.edu`

[3] The command-line options for SVD is `-a ranksvd exp_add 5 5 25`.

[4] The command-line options for LCR is `-a pgllorma exp_add 5 5 13`.

[5] `https://github.com/dhpark22/collranking`

[6] `http://www.vlfeat.org/`

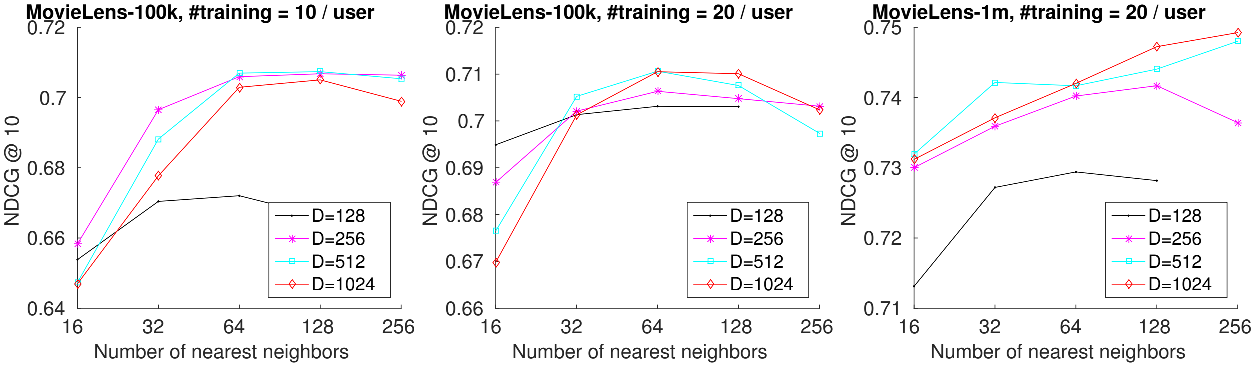

Figure 1: Ranking quality NDCG@10 (Section 6.3) as a function of the number of nearest neighbors (k) when constructing the *kd*-tree and the maximum number of comparisons (D) when querying the *kd*-tree. k must not be larger than D.

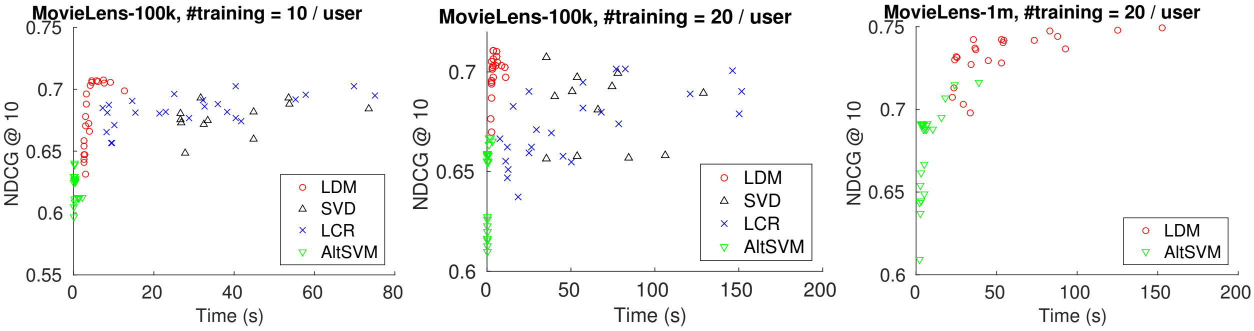

Figure 2: Ranking quality NDCG@10 (Section 6.3) vs. run-time under various hyperparameters for each of the methods compared. LCR and SVD are time-consuming and therefore their results with multiple sets of hyperparameters are not available on MovieLens-1m.

We only evaluate NDCG@10 due to limited space. Note that for a user u, the list of items that is provided to compute (19) is *all* the items with observed ratings given by u in the test set, not only the highest-ranking ones.

In contrast to previous work on top-N recommender systems [8, 5], we discourage the use of Precision@K in the context of collaborative ranking for recommender systems, which measures the proportion of actually rated items out of the top K items in the ranked list of all the items in the data set. Contemporary recommender systems typically use numeric ratings rather than binary ratings. In a 1-to-5 rating system, for example, a 1-star item should be less favorable than an unrated item with an expected 3-star rating. However, a recommender system that ranks 1-star items at the top positions would get a higher Precision@K score than one that ranks unrated items at the top positions. Thus, Precision@K is not a valid measure for collaborative ranking with a non-binary rating system.

6.4 Effect of Parameter Selection

First, we examine the influence of the *kd*-tree parameters on the performance of LDM, namely the number of nearest neighbors k and the maximum number of distance comparisons D. Fig. 1 shows the change in NDCG@10 when varying k and D on several small data sets (due to time constraints). In general, the ranking quality is much better with moderately large k, D values

Table 2: Benchmarking results of ranking quality NDCG@10 (Section 6.3) and run-time for all the compared methods. N is the number of ratings in the training set for each user.

	NDCG@10				**Time (seconds)**			
	SVD	LCR	AltSVM	LDM	SVD	LCR	AltSVM	LDM
MovieLens-1m, $N = 10$	0.6836	0.7447	0.6680	0.7295	844.4	254.2	3.6	61.1
MovieLens-1m, $N = 20$	0.6758	0.7428	0.6879	0.7404	843.3	437.3	6.8	52.3
MovieLens-1m, $N = 50$	0.6178	0.7470	0.7730	0.7527	730.5	1168.8	53.0	37.0
MovieLens-10m, $N = 10$	0.6291	0.6866	0.6536	0.7077	24913.4	4544.4	61.2	1496.3
MovieLens-10m, $N = 20$	0.6201	0.6899	0.7208	0.7213	14778.5	6823.5	275.4	1653.1
MovieLens-10m, $N = 50$	0.5731	0.6830	0.7507	0.7286	10899.1	14668.5	648.4	1295.0
MovieLens-10m, $N = 100$	0.5328	0.7125	0.7719	0.7349	14648.5	4289.1	1411.2	832.0

than with very small k, D values, but does not improve much when further increasing k and D. Therefore, we can use sufficiently large k, D values to get good ranking quality, but not too large to be computationally efficient. In the experimental comparison in Section 6.5, we fix the parameters to $k = 64$ and $D = 256$.

Next, we vary the hyperparameters in each of the four methods, and compare simultaneously the ranking quality and run-time under different hyperparameters. Ideally, a good performance of a collaborative ranking method means producing higher NDCG@10 scores in less time. Fig. 2 plots NDCG@10 against the run-time for several small data sets (due to time constraints). LDM achieves the highest NDCG@10 in a reasonable amount of time compared to the other methods. AltSVM is efficient but produces unsatisfactory ranking quality, which is also sensitive to its hyperparameters. For LCR, the ranking quality is acceptable but it takes considerably longer time, especially when the size of training set increases. On MovieLens-100k ($N = 20$), SVD and LDM achieve similar NDCG@10 scores but LDM costs much shorter run-time.

6.5 Results

Now we fix the hyperparameters as described in Section 6.2. Table 2 reports the run-time and NDCG@10 scores for all the compared methods on the larger data sets. LDM does not achieve the highest NDCG@10 scores in every case, but produces robust ranking quality with decent run-time (except MovieLens-10m, $N = 50$). For LCR and SVD, the time cost increases dramatically on larger data sets. AltSVM achieves superior ranking quality when the number of training ratings N is large, but its performance is sensitive to the number of iterations, which in turn depends on the data set and the given tolerance parameter. We conclude that LDM is an overall competitive method that is efficient and robust to hyperparameters and the underlying data sets. Also, LDM has particular advantages when the available information is relatively few, i.e. when N is small, which we consider is a more difficult problem than the cases with richer training information.

We can clearly see that the run-time of LDM increases with the number of users (due to the reliance on the user-user similarity matrix, as expected), while the run-time of AltSVM increases with the number of ratings in the training set. Further optimizing the code for LDM is an important direction in our future work to make it efficient for large-scale data sets.

We note that our method predicts all the missing ratings in the label propagation process, which is included in the timing of LDM, and therefore our methods takes a negligible amount of time for testing, especially in the real scenario where a recommendation algorithm would have to predict the ratings for all the items and return the top-rated ones to the user.

7 Related Work

Both user-based and item-based collaborative filtering [4, 18] can be considered as graph-based local label propagation methods. The idea of discrete harmonic extension for semi-supervised learning in general was originally proposed in [24]. Graph-based methods were recently considered for matrix completion [7] and top-N recommendation [23] as well. Given all the existing work, what we have presented is a continuous harmonic extension formulation for label propagation and a rigorous manifold learning algorithm for collaborative ranking.

8 Conclusion and Discussion

In this paper, we have proposed a novel perspective on graph-based methods for matrix completion and collaborative ranking. For each item, we view the user-user graph as a partially labeled data set (or vice versa), and our method propagates the known labels to the unlabeled graph nodes through the graph edges. The continuous harmonic extension problem associated with the above semi-supervised learning problem is defined on a user or item manifold solved by a point integral method. Our formulation can be seen as minimizing the dimension of the user or item manifold, and thus builds a smooth model for the users or items but with higher complexity than low-rank matrix approximation. Also, our method can be fully parallelized on distributed machines, since the linear systems that need to be solved for all the items are independent with one another. Experimental results have shown that our method has particular strength when the number of available ratings in the training set is small, which makes it promising when combined with other state-of-the-art methods like AltSVM. Our formulation for harmonic extension in the context of matrix completion can be extended to include other constraints or regularization terms and side information as well. An important direction is to further improve the efficiency of the algorithm to be comparable with recent large-scale matrix completion methods [22, 25].

References

[1] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *Commun. ACM*, vol. 51, no. 1, pp. 117–122, 2008.

[2] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[3] D. Billsus and M. J. Pazzani, "Learning collaborative information filters," in *ICML '98: Proc. of the 15th Int. Conf. on Machine learning*, 1998, pp. 46–54.

[4] J. S. Breese, D. Heckerman, and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in *UAI '98: Proc. of the 14th Conf. on Uncertainty in Artificial Intelligence*, 1998, pp. 43–52.

[5] M. Deshpande and G. Karypis, "Item-based top-n recommendation algorithms," *ACM Trans. Inf. Syst.*, vol. 22, no. 1, pp. 143–177, 2004.

[6] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of IR techniques," *ACM Trans. Inf. Syst.*, vol. 20, no. 4, pp. 422–446, 2002.

[7] V. Kalofolias, X. Bresson, M. Bronstein, and P. Vandergheynst, "Matrix completion on graphs," in *NIPS Workshop on "Out of the Box: Robustness in High Dimension"*, 2014.

[8] G. Karypis, "Evaluation of item-based top-n recommendation algorithms," in *CIKM '01: Proc. of the 10th Int. Conf. on Information and Knowledge Management*, 2001, pp. 247–254.

[9] J. Lee, S. Bengio, S. Kim, G. Lebanon, and Y. Singer, "Local collaborative ranking," in *WWW '14: Proc. of the 23rd Int. Conf. on World Wide Web*, 2014, pp. 85–96.

[10] J. Lee, S. Kim, G. Lebanon, and Y. Singer, "Local low-rank matrix approximation," in *ICML '13: Proc. of the 30th Int. Conf. on Machine learning*, 2013.

[11] J. Lee, M. Sun, and G. Lebanon, "A comparative study of collaborative filtering algorithms," 2012, http://arxiv.org/abs/1205.3193.

[12] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *VISAPP: Proc. of the Int. Conf. on Computer Vision Theory and Applications*, 2009, pp. 331–340.

[13] X. Ning and G. Karypis, "Recent advances in recommender systems and future directions," in *Pattern Recognition and Machine Intelligence*, ser. Lecture Notes in Computer Science. Springer, 2015, vol. 9124, pp. 3–9.

[14] S. O'Hara and B. A. Draper, "Are you using the right approximate nearest neighbor algorithm?" in *WACV '13: Workshop on Applications of Computer Vision*, 2013.

[15] S. Osher, M. Burger, D. Goldfarb, J. Xu, and W. Yin, "An iterative regularization method for total variation-based image restoration," *Multiscale Modeling & Simulation*, vol. 4, no. 2, pp. 460–489, 2005.

[16] S. Osher, Z. Shi, and W. Zhu, "Low dimensional manifold model for image processing," UCLA, Tech. Rep. CAM report 16-04, 2016.

[17] D. Park, J. Neeman, J. Zhang, S. Sanghavi, and I. Dhillon, "Preference completion: Large-scale collaborative ranking from pairwise comparisons," in *ICML '15: Proc. of the 32th Int. Conf. on Machine learning*, 2015, pp. 1907–1916.

[18] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *WWW '01: Proc. of the 10th Int. Conf. on World Wide Web*, 2001, pp. 285–295.

[19] Z. Shi and J. Sun, "Convergence of the point integral method for the poisson equation on manifolds I: the Neumann boundary," 2014, http://arxiv.org/abs/1509.06458.

[20] Z. Shi, J. Sun, and M. Tian, "Harmonic extension," 2015, http://arxiv.org/abs/1509.06458.

[21] A. Vedaldi and B. Fulkerson, "VLFeat: An open and portable library of computer vision algorithms," 2008, http://www.vlfeat.org/.

[22] H. Yun, H.-F. Yu, C.-J. Hsieh, S. V. N. Vishwanathan, and I. Dhillon, "Nomad: Non-locking, stochastic multi-machine algorithm for asynchronous and decentralized matrix completion," *Proc. VLDB Endow.*, vol. 7, no. 11, pp. 975–986, 2014.

[23] Y. Zhang, J.-q. Wu, and Y.-t. Zhuang, "Random walk models for top-n recommendation task," *Journal of Zhejiang University SCIENCE A*, vol. 10, no. 7, pp. 927–936, 2009.

[24] X. Zhu, Z. Ghahramani, and J. Lafferty, “Semi-supervised learning using gaussian fields and harmonic functions,” in *ICML ’03: Proc. of the 20th Int. Conf. on Machine learning*, 2003, pp. 912–919.

[25] Y. Zhuang, W.-S. Chin, Y.-C. Juan, and C.-J. Lin, “A fast parallel sgd for matrix factorization in shared memory systems,” in *RecSys ’13: Proc. of the 7th ACM Conf. on Recommender Systems*, 2013, pp. 249–256.