A Kernel-Based Approach to Exploiting Interaction-Networks in Heterogeneous Information Sources for Improved Recommender Systems

Oluwasanmi Koyejo
Electrical and Computer Engineering Dept.
University Of Texas, Austin
sanmi.k@mail.utexas.edu

Joydeep Ghosh
Electrical and Computer Engineering Dept.
University Of Texas, Austin
ghosh@ece.utexas.edu

ABSTRACT

Pairwise interaction networks capture inter-user dependencies (e.g. social networks) and inter-item dependencies (e.g item categories) that provide insight into user and item behavior. It is often assumed that such interaction information is informative for preference prediction. This may not be the case, as the some of the observed interactions may not be correlated with the preferences, and their use may negatively impact performance by introducing undesired noise.

We propose an approach for weighting each interaction, such that we can determine the importance of each interaction to the preference prediction task. We model the preferences using kernel matrix factorization; where the kernels capture the weighted effects of the interactions. Our approach is validated on Last.fm and Movielens datasets; which include multiple sources of explicit and implicit inter-user and inter-item interactions. Our experiments suggest that learning the most important interactions can improve recommendation performance when compared to the standard matrix factorization approach.

1. INTRODUCTION

Consumers now expect certain information sources, such as news from Google[1], product recommendations from Amazon[2] and music recommendations from Last.fm[3] to be personalized to their unique tastes. One of the reasons for this phenomenon is the incredible growth of options available to the consumer. A recommender system becomes an information filter that helps the user to navigate the overwhelming breadth of choices. Recommender systems may also be used for targeted advertising and other personalized services.

There are several approaches to preference recommendation in practice [2]. Matrix factorization based models [11, 16] assume that the observed ratings may be explained by a small number of user and item factors. The user-item ratings are represented as a sparse matrix, and the model learns a low rank factorization that explains the data. In contrast, models such as SCOAL [6] also utilize available features. SCOAL decomposes the user-item matrix into co-clusters, and uses the features to learn local models for each co-cluster. The co-clusters recovered represent groups of users that are correlated in their preferences for clusters of related items.

The observation that social networks may influence user preferences is the basis for several recommender system models. Memory based systems predict user preferences by traversing the user graph [7], or the combined user/item graph [8]. The predicted user-item preference score is a weighted sum of the preferences of close neighbors on the same item (or closely related items). The weights are estimated using the length of the walk, the strength of the links, and the item similarities. Social networks may also be integrated with matrix factorization models. Ma et al. [13] jointly factorize the ratings matrix and the social network, with the constraint that both factorizations use the same user factors. A related approach [9] uses the graph to regularize on the user factors; so that the predicted factors vary smoothly over the social graph. One notable feature of these models is that they require the user to select special parameters that tune the effect of the social network. In contrast, our proposed model can automatically compute weights that parametrize this effect, avoiding potentially expensive cross validation.

Despite the focus on social networks, the interaction information available is often much richer. Users may interact in different contexts; such as in professional networks and in family relationships. In addition, the user may provide both implicit and explicit feedback of their preferences [10]. For example, the Last.fm dataset[4] includes explicit feedback of user-artist preferences, as represented by listening counts, but also contains tag information; representing the user action of tagging artists. Users that tag the same artists are likely to share musical tastes, so the action of tagging may be interpreted as an implicit indicator of user similarity with respect to artist preferences. Item similarity may be described in a similar manner. For example, movies that are tagged by the same set of users are similar with respect to the observed user tags. In addition to tag information, items are often described by categories. These categories may specify similarity based on expert opinions, or the collected opinions

[1] news.google.com
[2] www.amazon.com
[3] www.lastfm.com

[4] http://ir.ii.uam.es/hetrec2011/datasets/lastfm/readme.txt

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
HetRec '11, October 27, 2011, Chicago, IL, USA
Copyright 2011 ACM 978-1-4503-1027-7/11/10 ...$10.00.

of a user community. In the case of movies, the categories may include directors, actors and genres.

The interactions explain the correlations between the users and between the items. In turn, some of these interactions may be correlated with the user-item preferences, while others may be irrelevant. Ad-hoc methods of combining these interactions; such as averaging or hand tuning a weighted combination, are tedious and clearly sub-optimal. The use of irrelevant interactions may also introduce noise that may degrade prediction performance. In this paper, we propose an approach for learning recommendations using several heterogeneous data sources, each of which define pair-wise interactions. We will represent each of these interaction networks as graphs. We then represent each graph using a spectral decomposition. Our model learns weights for each of these sources based on their relevance to the prediction task.

Outline: We begin our discussion in section 2 by motivating the spectral representation, and discuss some relevant properties. We then review the matrix factorization model for recommender systems; generalized to use feature information (section 3). The high dimensionality of the space will motivate a kernelized matrix factorization approach; which we discuss in section 4. We will augment this model with a sparsity inducing regularizer for learning the most important interactions in section 4.1. In section 5, we propose an algorithm for estimating the model parameters, then we describe our experiments and results in section 6. We conclude in section 7 and include possible directions for future work.

Notation: $x_{i,j}$ denotes the $(i,j)^{th}$ entry of the matrix $\mathbf{X}$. $\mathbf{x}_{i,:}$ denotes the i^{th} row of $\mathbf{X}$ and $\mathbf{x}_{:,j}$ denotes the j^{th} column. Let Ω denote the index set (i,j) of observed entries in a matrix $\mathbf{X}$, with the number of observed entries, $|\Omega| = K$. We denote the partially observed matrix by $\mathbf{X}_\Omega$ with the un-observed entries set to 0. We also define the operation $\text{vec}_\Omega(\mathbf{X})$ that transforms a $M \times N$ matrix $\mathbf{X}$ into a vector $\mathbf{y}$ of size $K \times 1$ by stacking the columns and ignoring the entries $\{(i,j) \,|(i,j) \notin \Omega\}$. In the special case where $K = M \times N$, we will instead use the $\text{vec}(\cdot)$ operator which stacks the columns of a $M \times N$ matrix $\mathbf{X}$ into a vector $\mathbf{x}$ of size $(M \times N) \times 1$. The Kronecker product is denoted by $\otimes$.

2. INTERACTION GRAPHS AND FEATURE EXTRACTION

An interaction network is a description of the relationship between a group of entities. For example, a social network is an interaction network between users with links given by the social ties. A compact way of formally characterizing an interaction network is a graph[5] $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{A})$, where $\mathcal{V} = \{v_i\}$ represents the set of entities as vertices, $\mathcal{E} = \{e_{i,j}\}$ represents the set of links between the entities v_i and v_j, and the adjacency matrix $\mathbf{A}$ with entries $\{a_{i,j}\}$ representing the strength of these links. In a social network, the edges $e_{i,j}$ may indicate a relationship between two people, and the adjacency $a_{i,j}$ may represent the strength of those social ties. Without loss of generality, we will assume that all the weights $a_{i,j} > 0$. Further, we will focus on interactions where the links are symmetric i.e. $a_{i,j} = a_{j,i}$. The extension to non-symmetric links is left for future work.

We will be concerned with functions on the nodes of this network given by some $f : v_i \mapsto \mathbb{R}$. These functions, may describe, for example, the degree to which each person in a network enjoys classical music, or enjoys action movies. Here we make a modeling assumption; that the value of the function is correlated with the relationship between the entities. One way to constrain f to be correlated with the graph $\mathcal{G}$ is to enforce *smoothness* with respect to the graph. To achieve this, we will seek functions such that the average weighted square difference between function evaluations at adjacent vertices is small. In other words,

$$\sum_{i,j=1}^{|\mathcal{V}|} a_{i,j}(f(v_i) - f(v_j))^2 < C,$$

where C is some user defined constant. If we organize the function values into a vector $\boldsymbol{f} \in \mathbb{R}^{|\mathcal{V}|}$, so that $\boldsymbol{f}_i = f(v_i)$, one can show that this is equivalent to the constraint:

$$2\boldsymbol{f}^\top \mathbf{L} \boldsymbol{f} < C,$$

where $\mathbf{L}$ is the the graph *Laplacian* matrix of size $|\mathcal{V}| \times |\mathcal{V}|$. The Laplacian matrix can be computed as $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{A}$ is the graph adjacency matrix and $\mathbf{D}$ is a $|\mathcal{V}| \times |\mathcal{V}|$ diagonal matrix with $d_{i,i} = \sum_j a_{i,j}$. We may also define a normalized smoothness penalty:

$$\sum_{i,j=1}^{|\mathcal{V}|} a_{i,j} \left(\frac{f(v_i)}{\sqrt{d_{i,i}}} - \frac{f(v_j)}{\sqrt{d_{j,j}}} \right)^2 < C,$$

where the function values are are normalized by the weight densities at the vertices. This smoothness penalty is equivalent to constraint:

$$2\boldsymbol{f}^\top \widetilde{\mathbf{L}} \boldsymbol{f} < C,$$

using the *Normalized Laplacian* $\widetilde{\mathbf{L}} = \mathbf{D}^{-\frac{1}{2}} \mathbf{L} \mathbf{D}^{-\frac{1}{2}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}$. Several properties of these matrices are well understood [12, 19]. Both $\mathbf{L}$ and $\widetilde{\mathbf{L}}$ are symmetric positive definite matrices. The eigenvalues of $\widetilde{\mathbf{L}}$ are bounded as $0 \leq \lambda_i \leq 2$. The smallest eigenvalue of both graph Laplacians is 0, and the number of zero eigenvalues corresponds to the number of connected components in the graph. We note that these graph Laplacians can be used to generate an entire family of smoothness inducing penalties [19].

The eigenvectors of the graph Laplacians form an orthonormal basis for all smooth functions on the graph i.e. functions evaluated at the vertices. Therefore, any function that is *smooth* with respect to the graph may be represented at the node vertices as:

$$f(v_i) = \sum_j w_j \boldsymbol{\eta}_j(v_i).$$

Thus, the problem of learning a smooth function on the graph is reduced to learning the weight vectors w_j. This basis can be arranged in order of *smoothness*; with eigenvector of the smallest eigenvalue corresponding to the smoothest function, and larger eigenvalues corresponding to rougher functions. The eigenvectors corresponding to the D smallest eigenvalues contain most of the smoothness information we are interested in capturing. Therefore, for computational

[5]Though we will focus on graphs directly induced by interactions, we may describe feature variables using a similar approach. Given two points x_i and x_j, and a distance measure $d(x_i, x_j)$, a graph can be defined by connecting any nodes with $d(x_i, x_j) < \epsilon$ by an edge with weight inversely proportional to any monotonic function of their inter-point distance. Such derived graphs are used in semi-supervised learning, and have close connections to learning the manifold of a set of data points [4].

reasons, we will restrict our basis to the D smallest eigenvectors of the Laplacian matrix.

3. MATRIX FACTORIZATION

The goal of matrix factorization(MF) is to decompose the observed preferences into hidden user and item factors [11, 16]. In a movie recommendation application, the user factors may define a particular user's preference for action movies, exotic locations, and so on. In turn, the corresponding dimensions in the movie factor may define an *action score*, an *exotic location score* and other relevant metrics. The inner product between these factors defines the user-movie preference score as a measure of the alignment between the user interests and the movie characteristics. We note that these factors are not predefined, and must be learned by the model. In practice, the recovered factors are often abstract and are difficult to interpret directly.

We assume that there is an available training set of N observations $\{(i,j) \in \Omega\}$, where $z_{i,j}$ be the observed preference of the i^{th} user for the j^{th} item. Let $\mathbf{x}_i \in \mathbb{R}^{D_x}$ be the feature vector for each user i, and $\mathbf{y}_j \in \mathbb{R}^{D_y}$ for each item j. We can combine both features to define a feature vector for the $(i,j)^{th}$ example. In this paper, we will use the Kronecker product as a combiner i.e. we define the $(i,j)^{th}$ feature vector as $\mathbf{x}_i \otimes \mathbf{y}_j \in \mathbb{R}^{D_x \times D_y}$. We will use a linear predictor; which involves estimating a parameter vector $\mathbf{w} \in \mathbb{R}^{D_x \times D_y}$ so the resulting predictions are:

$$\begin{aligned} \hat{z}_{i,j} &= \mathbf{w}^\top (\mathbf{x}_i \otimes \mathbf{y}_j) \\ &= \mathbf{x}_i^\top \mathbf{W} \mathbf{y}_j. \end{aligned} \tag{1}$$

where $\mathbf{w} = \text{vec}(\mathbf{W})$. The result (1) follows by an application of the standard Kronecker identity.

In the high dimensional setting, the parameter space of $\mathbf{W} \in \mathbb{R}^{D_x \times D_y}$ may be too large to learn an accurate model, so we will restrict the space of parameters by enforcing some maximum rank R. Now $\mathbf{W}$ may be parametrized using a factorized form $\mathbf{W} = \mathbf{U}\mathbf{V}^\top$ where $\mathbf{U} \in \mathbb{R}^{D_x \times R}$ and $\mathbf{V} \in \mathbb{R}^{D_y \times R}$. The prediction is now computed as:

$$\hat{z}_{i,j} = \mathbf{x}_i^\top \mathbf{U} \mathbf{V}^\top \mathbf{y}_j, \tag{2}$$

with the user factors given by the linear term $\mathbf{x}_i^\top \mathbf{U}$ and the item factors given by $\mathbf{y}_i^\top \mathbf{V}$.

Standard matrix factorization [11, 16] is a special case of this model where the user (resp. item) features are categorical, specifying a unique identifier for each user (resp. item). Without loss of generality, we may represent user i using the feature $\mathbf{x}_i = \mathbf{e}_i \in \mathbb{R}^{N_x}$ where $\mathbf{e}_i$ is the i^{th} standard basis vector, with 1 at the i^{th} dimension, and 0 elsewhere. Similarly, item j is represented by the feature $\mathbf{y}_j = \mathbf{e}_j \in \mathbb{R}^{N_y}$. In matrix form, the predicted matrix is given by the model, $\widehat{\mathbf{Z}} = (\mathbf{IU})(\mathbf{VI})^\top = \mathbf{U}\mathbf{V}^\top$. Note that in standard matrix factorization, the linear model in (1) without the low rank constraint severely overfits the data and the low rank constraint is critical for any generalization. In fact, if we use the squared loss function as our error measure, the least square estimate of the weight vector predicts the training data exactly as $\hat{z}_{i,j} = w_{i,j} = z_{i,j}$ for every $(i,j) \in \Omega$ and predicts $\hat{z}_{i,j} = 0$ everywhere else.

The basis vectors $\mathbf{e}_i$ are also a basis for functions on the graph $\mathcal{G}$, but they do not enforce smoothness, as each vertex is not constrained in its interaction with its neighbors. To include the smoothness effect, we will augment the identity features with the graph eigenvector features described in section 2. Given some graph $\mathcal{G}_x$ over the users and $\mathcal{G}_y$ over the items, we will extract the eigenvectors of the respective graph Laplacian matrices to compute new features $\boldsymbol{\eta}_x \in \mathbb{R}^{N_x \times D_x}$ and $\boldsymbol{\eta}_y \in \mathbb{R}^{N_y \times D_y}$. The new features are given by concatenation $\mathbf{x}_i{}^\top = [\,\mathbf{e}_i{}^\top, \boldsymbol{\eta}_x(i)^\top] \in \mathbb{R}^{1 \times (N_x + D_x)}$ and $\mathbf{y}_j{}^\top = [\,\mathbf{e}_j{}^\top, \boldsymbol{\eta}_y(j)^\top] \in \mathbb{R}^{1 \times (N_y + D_y)}$ respectively. Further, given different graphs from several user interaction networks $\{\mathcal{G}_x^k\}$ and several item interaction networks $\{\mathcal{G}_y^l\}$, we can create a feature space representation using the same procedure, with feature vectors given by $\mathbf{x}_i{}^\top = [\mathbf{e}_i{}^\top, \boldsymbol{\eta}_x^1(i)^\top, \cdots, \boldsymbol{\eta}_x^K(i)^\top] \in \mathbb{R}^{1 \times (N_x + \sum_k D_x^k)}$ for the user side and $\mathbf{y}_j{}^\top = [\,\mathbf{e}_j{}^\top, \boldsymbol{\eta}_y^1(j)^\top, \cdots, \boldsymbol{\eta}_y^L(j)^\top] \in \mathbb{R}^{1 \times (N_y + \sum_l D_y^l)}$ for the items. The result is a function that is *smooth* with respect to all the interaction graphs simultaneously. As we continue to include more interaction networks, the dimension of this space can grow very large. In such problems, where $D \gg N$, we may obtain a more efficient representation by using kernel functions.

4. KERNEL MATRIX FACTORIZATION

We will now extend our representation to functions in a reproducing kernel Hilbert space $f(x,y) \in \mathcal{F}$. Linear functions in $\mathcal{F}$ are given by given by $f(x,y) = \mathbf{w}^\top \boldsymbol{\phi}(x,y)$, using basis functions $\boldsymbol{\phi}(\cdot) \in \mathcal{F}$. Defining the function using this basis function representation may be unwieldy, when, for example $\boldsymbol{\phi}(\cdot)$ is an infinite dimensional representation. Instead, we will define a kernel function $s((x,y),(\hat{x},\hat{y})) = \langle \boldsymbol{\phi}(x,y), \boldsymbol{\phi}(\hat{x},\hat{y}) \rangle$ [17] so that functions may be evaluated as $f(x,y) = \sum_{n=1}^{\infty} m_n s((x,y),(x_n,y_n))$. The kernel function measures the pair-wise similarity between examples and must be positive semi-definite. Given some convex loss function $\mathcal{L}(z_n, f(x_n,y_n))$, and the induced norm $\|f\|_{\mathcal{F}}^2 = \langle f, f \rangle$, the function that minimizes the regularized cost is given by $f(x,y) = \sum_{n=1}^{N} m_n s((x,y),(x_n,y_n))$. In this way, the infinite dimensional optimization over the function space $\mathcal{F}$ is converted to a tractable finite dimensional optimization over the set of parameters $\{m_n\}$.

Let the space of functions over the users $\mathcal{U}$ be defined by some kernel $k(\cdot,\cdot)$, and the space of functions over the items $\mathcal{V}$ be defined by $g(\cdot,\cdot)$. We will define functions in the Hilbert product space $\mathcal{F} = \mathcal{U} \times \mathcal{V}$ using the product kernel $s((x,y),(\hat{x},\hat{y})) = k(x,\hat{x})g(y,\hat{y})$. The kernel model suffers from a similar explosion in dimensionality as the linear model we described in (1), and we will again enforce a rank constraint. The kernel function with $\text{rank}(f) < R$ can be represented using the sum $f = \sum_{r=1}^{R} f_r$, where f_r are rank one functions[6]. We will regularize the space using the pseudo-norm $\|f\|_{\mathcal{F}}^2 = \|\mathbf{U}\|_{\mathcal{H}_K}^2 + \|\mathbf{V}\|_{\mathcal{H}_G}^2$. The proof that this function space satisfies the representer theorem follows directly from [1, 17] and is omitted[7]. Each rank one function

[6]The rank R of a function $f \in \mathcal{F}$ is defined as the minimum number of rank one atomic terms f_r that can be used to define f [1, Appendix A]. If R does not exist, then $\text{rank}(f) = \infty$.

[7]As in the matrix case [15], the pseudo-norm using the factors is closely related to nuclear norm of the kernel function[1]; often used as a heuristic for rank minimization. $\|f\|_* = \inf_{f = \sum_{r=1}^{\infty} u_r \otimes v_r} \frac{1}{2} \sum_{r=1}^{\infty} \left(\|u_r\|_{\mathcal{H}_K}^2 + \|v_r\|_{\mathcal{H}_G}^2 \right)$

f_r is given by:

$$f_r(x,y) = \sum_{i=1}^{\infty} k(x,x_i)\alpha_r^i \sum_{j=1}^{\infty} g(y,y_j)\beta_r^j.$$

We now apply the representer theorem [17] to compute the predictions as:

$$\hat{z}_{i,j} = \mathbf{k}_i \boldsymbol{\alpha}\boldsymbol{\beta}^\top \mathbf{g}_j \tag{3}$$

where $\boldsymbol{\alpha} \in \mathbb{R}^{N_u \times R}$, $\boldsymbol{\beta} \in \mathbb{R}^{N_m \times R}$, $\mathbf{k}_i(l) = k(x_i, x_l)$ and $\mathbf{g}_j(l) = k(y_j, y_l)$. For comparison with matrix factorization, let $\mathbf{u}_i = \mathbf{k}_i\boldsymbol{\alpha} \in \mathbb{R}^R$ and $\mathbf{v}_j = \mathbf{g}_j\boldsymbol{\beta} \in \mathbb{R}^R$, then the predicted function can be written in the familiar form; $\hat{z}_{i,j} = \mathbf{u}_i^\top \mathbf{v}_j$. The kernel values for users and items not in the training set may be estimated using the Nyström approximation [5]. This is especially relevant when the number of users or items is large, and the cost of repeated eigen-decompositions is prohibitive.

4.1 Sparse Kernel weights

Our goal is to develop a model that can learn the importance of each interaction. For this reason, we will decompose the kernel representation into separate sub-kernels; each capturing a different basis extracted from the graphs. First, we note that if $\mathbf{k}(i,j) = \langle \boldsymbol{\phi}(i), \boldsymbol{\phi}(j)\rangle$ is the kernel corresponding to some feature space $\boldsymbol{\phi}(\cdot)$, then the concatenated feature space $\boldsymbol{\phi}(\cdot)^\top = [\,\boldsymbol{\phi}_0(\cdot)^\top, \boldsymbol{\phi}_1(\cdot)^\top]$ has the kernel space representation $k(i,j) = k_0(i,j) + k_1(i,j)$ where $k_0(i,j) = \langle \boldsymbol{\phi}_0(i), \boldsymbol{\phi}_0(j)\rangle$ and $k_1(i,j) = \langle \boldsymbol{\phi}_1(i), \boldsymbol{\phi}_1(j)\rangle$.

Let $\mathbf{x}_i{}^\top = [\,\mathbf{e}_i{}^\top, \boldsymbol{\eta}_x^1(i)^\top, \cdots, \boldsymbol{\eta}_x^K(i)^\top]$ as discussed in section 3. We may concatenate these row vectors into $\mathbf{X} \in \mathbb{R}^{N_x \times (N_x + \sum_k D_x^k)}$ such that $\mathbf{X}_{i,:} = \mathbf{x}_i{}^\top$. We define the map $\pi : p \mapsto (\mathcal{G}_k, d)$ so that p corresponds to the d^{th} eigenvectors of the k^{th} graph. Now we can decompose the kernels as $\mathbf{K} = \sum_{p=0}^{P} \mathbf{K}_p$. $\mathbf{K}_0 = \mathbf{I}$ captures the self-correlation at each vertex which is independent of all other nodes. The other kernels are computed as $\mathbf{K}_p = \mathbf{X}_{:,N_x+p}\mathbf{X}_{:,N_x+p}{}^\top$ for $p > 0$. The *eigen-kernel* $\mathbf{K}_p$ for $p > 0$ captures the eigenvector feature $\boldsymbol{\eta}_{x,d}^k$ (using the map $\pi(p) = (\mathcal{G}_k, d)$). The same approach may be applied to define item kernels where $\mathbf{Y}_{i,:} = \mathbf{y}_i{}^\top$ and $\mathbf{Y} \in \mathbb{R}^{N_y \times (N_y + \sum_l D_y^l)}$. Now $\mathbf{G} = \sum_{q=0}^{Q} \mathbf{G}_q$ where $\mathbf{G}_0 = \mathbf{I}$ and $\mathbf{G}_q = \mathbf{Y}_{:,N_y+q}\mathbf{Y}_{:,N_y+q}{}^\top$ for $q > 0$. The sub-kernels $\mathbf{K}_p$ and $\mathbf{G}_q$ induced by the eigenvectors can be interpreted as sub-correlations between the node vertices. Such correlations capture local smoothness between the graph vertices based on the inter-user and inter-item interactions. This smoothness is closely related to local clustering in the graph nodes; as a smooth function will be almost constant over connected clusters.

We now introduce kernel weights a_p and b_q to parametrize the influence of each sub-kernel, so that:

$$\mathbf{K} = \sum_{p=1}^{P} a_p \mathbf{K}_p \;\text{ and }\; \mathbf{G} = \sum_{q=1}^{Q} a_q \mathbf{G}_q.$$

These weights are constrained to lie in the simplex $\{a_p \geq 0, \sum_p a_p = 1\}$ and $\{b_q \geq 0, \sum_q b_q = 1\}$. The simplex constraint is closely related to a lasso penalty, as the contours of this set encourage sparsity. These weights may also be interpreted as probabilities. The kernels must be normalized in order to recover meaningful solutions. We will normalize all kernels to unit trace. Learning a sparse combination of kernels is closely related to multiple kernel learning[14], so our model may be interpreted as a combination of multiple kernel learning with kernel matrix factorization.

The weights separate the model into interpretable factors; based on the different interaction effects between the graph kernels on the predicted ratings. We may compute the prediction for the entire matrix as $\widehat{\mathbf{Z}} = \mathbf{K}\boldsymbol{\alpha}\boldsymbol{\beta}^\top\mathbf{G}$; decomposed as $\sum_{p=1}^{P}\sum_{q=1}^{Q} a_p b_q \mathbf{K}_p\boldsymbol{\alpha}\boldsymbol{\beta}^\top\mathbf{G}_q$. The learned factors can be expressed as a weighted linear combination of a collaborative filtering term: $a_0 b_0 \boldsymbol{\alpha}\boldsymbol{\beta}^\top$ which captures self correlations not explained by the features, interaction between the user ratings and item kernels, $\sum_{q=1}^{Q} a_0 b_q \boldsymbol{\alpha}\boldsymbol{\beta}^\top\mathbf{G}_q$, interaction between the item entries and the user kernels, $\sum_{p=1}^{P} a_p b_0 \mathbf{K}_p\boldsymbol{\alpha}\boldsymbol{\beta}^\top$, and the pair-wise interaction between the different user kernels and item kernels, $a_p b_q \mathbf{K}_p\boldsymbol{\alpha}\boldsymbol{\beta}^\top\mathbf{G}_q$ for $p > 0$ and $q > 0$.

Related Work: We have limited our discussion to the use of symmetric graphs. In contrast, regularization based matrix factorization models such as [9] may not require this symmetry. If the graph is symmetric and only the user social network graph is used, the model in [9] may be recovered using the equivalence between regularization and kernels [19]. Our model may then be seen as as extension of [9] with multiple inter-user and inter-item interactions. The idea of using the graph eigenvectors as features for was explored in [18] to improve classification performance. Argyriou et al. [3] explored the use of multiple kernel techniques to learn a sum of Laplacian kernels. In contrast, we model the kernel sparsity at the more granular eigenvector level.

5. PARAMETER ESTIMATION

We measure model error using the least squares loss; computed as $l(z_i, f(x_i, y_i)) = (z_i - f(x_i, y_i))^2$. This is combined with the kernel regularizer to define the regularized cost:

$$C = \frac{1}{2}\left\|\left[\mathbf{K}\boldsymbol{\alpha}\boldsymbol{\beta}^\top\mathbf{G} - \mathbf{Z}\right]_\Omega\right\|_F^2 + \frac{\lambda}{2}\left(tr(\boldsymbol{\alpha}^\top\mathbf{K}\boldsymbol{\alpha}) + tr(\boldsymbol{\beta}^\top\mathbf{G}\boldsymbol{\beta})\right) \tag{4}$$

We will use the representations $\mathbf{U} = \mathbf{K}\boldsymbol{\alpha}$ and $\mathbf{V} = \mathbf{G}\boldsymbol{\beta}$ as appropriate to simplify the notation. The cost function in terms of the entire set of parameters $\{\boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{a}, \mathbf{b}\}$ is non-convex. However, if all but one of the parameters is fixed, the cost function in terms of the free parameter becomes convex, and in fact, each sub-optimization problem reduces to a system of linear equations. This motivates a simple coordinate descent algorithm to find a local minimum of the cost.

We begin by learning the user factor variable $\boldsymbol{\alpha}$. If $\boldsymbol{\beta}$, $\mathbf{a}$ and $\mathbf{b}$ are fixed, we may estimate $\boldsymbol{\alpha}$ by solving the following regularized least squares problem:

$$\frac{1}{2}\left\|\left[\mathbf{K}\boldsymbol{\alpha}\mathbf{V}^\top - \mathbf{Z}\right]_\Omega\right\|_F^2 + \frac{\lambda}{2} tr(\boldsymbol{\alpha}^\top\mathbf{K}\boldsymbol{\alpha}) \tag{5}$$

where $tr(\mathbf{M})$ denotes the trace of some matrix $\mathbf{M}$. The gradient is given by:

$$\frac{\partial C}{\partial \boldsymbol{\alpha}} = \mathbf{K}\mathbf{E}\mathbf{V} + \lambda\mathbf{K}\boldsymbol{\alpha}$$

where the sparse error matrix $\mathbf{E} = \left[\mathbf{K}\boldsymbol{\alpha}\boldsymbol{\beta}^\top - \mathbf{Z}\right]_\Omega$. Similarly, when $\boldsymbol{\alpha}$, $\mathbf{a}$ and $\mathbf{b}$ are fixed, we may learn the item factor $\boldsymbol{\beta}$ by solving:

$$\frac{1}{2}\left\|\left[\mathbf{G}\boldsymbol{\beta}\mathbf{U}^\top - \mathbf{Z}^\top\right]_\Omega\right\|_F^2 + \frac{\lambda}{2} tr(\boldsymbol{\beta}^\top\mathbf{G}\boldsymbol{\beta}) \tag{6}$$

Algorithm 1 Co-ordinate Descent Solver

1: **initialize** $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\mathbf{a}$ and $\mathbf{b}$
2: **repeat**
3: Compute $\mathbf{a}^*$ by solving (7)
4: Compute $\boldsymbol{\alpha}^*$ by solving (5)
5: **repeat** ▷ Optional inner loop
6: re-compute $\mathbf{a}^*$ by solving (7)
7: re-compute $\boldsymbol{\alpha}^*$ by solving (5)
8: **until** Converged
9: Compute $\mathbf{b}^*$ by solving (8)
10: Compute $\boldsymbol{\beta}^*$ by solving (6)
11: **repeat** ▷ Optional inner loop
12: re-compute $\mathbf{b}^*$ by solving (8)
13: re-compute $\boldsymbol{\beta}^*$ by solving (6)
14: **until** Converged
15: **until** Converged
16: **return** $\boldsymbol{\alpha}^*$, $\boldsymbol{\beta}^*$, $\mathbf{a}*$ and $\mathbf{b}^*$

with the gradient given by:

$$\frac{\partial C}{\partial \boldsymbol{\beta}} = \mathbf{G}\mathbf{E}^\top \mathbf{U} + \lambda \mathbf{G}\boldsymbol{\beta}$$

We solve the linear systems for $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ using the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm. Note that, as in most matrix factorization models, the point $(\boldsymbol{\alpha}, \boldsymbol{\beta}) = (\mathbf{0}, \mathbf{0})$ is a local minimum of (4), and so $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ must be initialized to non-zero values. We initialized $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ using random values.

The solution for kernel weight parameters once again corresponds to a quadratic problem, but now with convex constraints. Let $\mathbf{z} = \text{vec}_\Omega(\mathbf{Z})$ and $\mathbf{S}_{:,p} = \text{vec}_\Omega\left(\mathbf{K}_p \boldsymbol{\alpha} \mathbf{V}^\top\right)$ i.e the predicted function learned using only one of the user kernels, so $\mathbf{S} \in \mathbb{R}^{N \times (P+1)}$. Let the individual function norms be given by $\mathbf{n}_p = tr(\boldsymbol{\alpha}\mathbf{K}_p\boldsymbol{\alpha})$. With $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and $\mathbf{b}$ fixed, the cost function with respect to $\mathbf{a}$ is the regularized least squares problem:

$$\frac{1}{2}\left\|\mathbf{z} - \mathbf{S}\mathbf{a}\right\|_2^2 + \frac{\lambda}{2}\mathbf{n}^\top \mathbf{a} \tag{7}$$
$$\text{s.t. } a_p \geq 0, \ \sum_{p=0}^{P} a_p = 1$$

Similarly, define $\mathbf{T}_{:,q} = \text{vec}_\Omega\left(\mathbf{G}_q \boldsymbol{\beta} \mathbf{U}^\top\right)$ and $\mathbf{m}_q = tr(\boldsymbol{\beta}\mathbf{G}_q\boldsymbol{\beta})$. If we fix $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and $\mathbf{a}$, the cost for $\mathbf{b}$ may be computed as:

$$\frac{1}{2}\left\|\mathbf{z} - \mathbf{T}\mathbf{b}\right\|_2^2 + \frac{\lambda}{2}\mathbf{m}^\top \mathbf{b} \tag{8}$$
$$\text{s.t. } b_q \geq 0, \ \sum_{q=0}^{Q} b_q = 1$$

We solve for $\mathbf{a}$ and $\mathbf{b}$ using a constrained quadratic problem solver. $\mathbf{a}$ and $\mathbf{b}$ are initialized as uniform vectors $a_i = 1/(P+1)$ and $b_j = 1/(Q+1)$.

6. EXPERIMENTS

We tested our model using the Last.fm and the Movielens datasets[8]. The data is pre-processed by removing global bias effects; computed using an un-regularized global linear model. All models are trained on the residual of the **global model**. We compared the **weighted interaction** model to **matrix factorization (MF)** and the **uniform** kernel using uniform weights $a_p = 1/(P+1)$ and all $b_q = 1/(Q+1)$. Note that standard matrix factorization is a special case of our model where $a_p = 1$ if $p = 0$ and $a_p = 0$ otherwise, and, $b_q = 1$ if $q = 0$ and $b_q = 0$ otherwise. We experimented with rank 5 and rank 10 factorization models, and present results on both. Our model includes a single regularization parameter λ. We selected λ from the set $\{10^{-5}, 10^{-4}, 5 \times 10^{-4}, 10^{-3}, 5 \times 10^{-3}\}$. We did not implement the optional inner loop in Algorithm 1 (lines 5 and 11).

We used the normalized Laplacian as our graph representation. Several of our graphs are generated using implicit interactions. For example, the pair-wise user similarity graph based on shared tags is extracted from the the user-item tag graph. This symmetry leads to some redundancy that we may exploit to reduce the computational overhead of computing the spectral representation. Let $\mathbf{M}$ be the adjacency matrix of the implicit graph. Then the interaction is described by the adjacency matrix $\mathbf{A} = \mathbf{M}\mathbf{M}^\top$. Fortunately, the symmetric graph Laplacian shares the same eigenvector space as the the weighted adjacency matrix. In other words, the eigenvectors of $\widetilde{\mathbf{L}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{M}\mathbf{M}'\mathbf{D}^{\frac{1}{2}}$ are identical to the left singular values of $\mathbf{D}^{-\frac{1}{2}}\mathbf{M}$, with eigenvalues given by $1 - \sigma_i^2$; where σ_i are the singular of $\mathbf{D}^{-\frac{1}{2}}\mathbf{M}$. Using this identity, we may avoid computing the these interactions explicitly, and avoid the factorization of a potentially large and dense graph.

6.1 Last.fm

The Last.fm task involves predicting the artist listening preferences of users. The information available in the dataset includes a social network between users and a set of user-artist tags. There are $N_x = 1982$ users and $N_y = 17632$ items, with $N = 92834$ user-listened artist counts. This results in a user-artist matrix with a density of 0.28%. Our hypothesis is that the number of times the user listens to an artist is an indicator of user preference. Fig. 1 shows the histogram of raw user counts with the probabilities on a log scale. Observe that the data is skewed with a long tail. We log-transformed the counts of user listen counts. Figure Fig. 2 shows the count distribution after log transformation. The data distribution is now closer to a Gaussian. Our results are reported using the log rescaled data as the target.

The user social network contains 25434 bi-directional links and 20 connected components. We extracted the first 50 eigenvalues of the normalized Laplacian to describe the social network. We also extracted the user-tag artist network. We ignored the tag counts and preserved only the presence/absence of a tag. We extracted the first 100 eigenvectors of the user-user tag graph, and the first 100 eigenvectors of the item-item tag graph. In all we extracted $P - 1 = 150$ user eigen-features and $Q - 1 = 100$ item eigen-features. The presence or absence of a user-artist tag, and a constant bias term were used to learn the initial global model.

We randomly divided the observed preferences into five sets, and present the 5-fold cross validation performance using root mean square error (RMSE) metric (see Table 6.1). The RMSE of the global model was comparable with the overall variance, suggesting that the global features were not informative. The uniform model performed only slightly better than the global model. Suggesting that the noise in the direct average overwhelmed the useful signals. Our model, which learns the weights of the interactions, out-

[8]http://www.grouplens.org/node/462

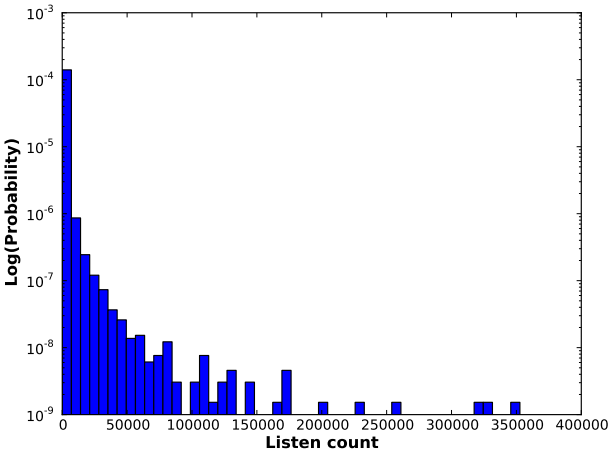

Figure 1: Histogram of raw listen counts in Last.fm $\mu = 745.24,\ \sigma = 3751.30$**.**

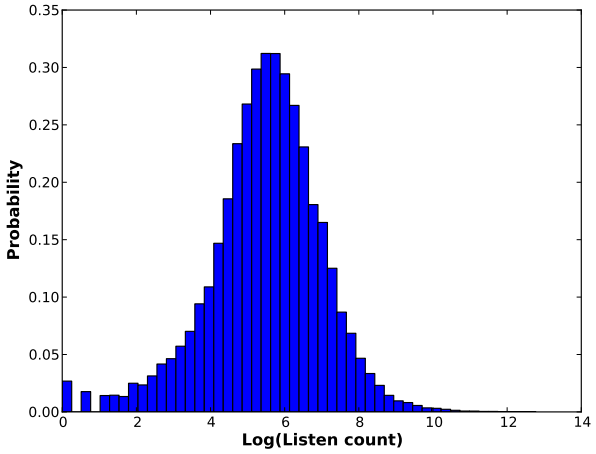

Figure 2: Histogram of log transformed listen counts in Last.fm $\mu = 5.469,\ \sigma = 1.531$**.**

performed plain matrix factorization. We observed that the rank 10 model slightly underperformed the rank 5 model; suggesting that the true rank is less than 10, and the rank 10 model may be over-fitting.

The models we learned were very sparse in the eigenbasis. In most of the runs, over 90% of the user kernel weights and the item kernel weights were zeros. We extracted the learned vectors and analyzed which of the features were most strongly correlated with the prediction. Fig. 3 and Fig. 4 show a representative result from the algorithm. The first eigen-kernel was selected as the most important user feature, and self-correlation was selected as the most important item feature. On the user side, less than 1% of the weight was assigned to the self-correlation, 93% of the weight was assigned to the social network, and 6% of the weight was assigned to the tag-based user similarity. On the item side, 77% of the weight was was assigned to self-correlation, and the item tag interactions cumulatively contributed 23% of the weight.

	Rank=5	rank=10
Global Model	1.502 (0.014)	-
Uniform Interaction	1.492 (0.015)	1.498 (0.008)
MF	1.139 (0.006)	1.173 (0.010)
Weighted Interaction	**1.071 (0.009)**	**1.106 (0.006)**

Table 1: Average (std.) cross validation RMSE on Last.fm

6.2 Movielens

The Movielens dataset contains user ratings expressing preferences for different movies. The dataset contains $N = 855598$ ratings. The meta-data available include user-movie tag information, movie genres, movie directors, country assignments, and aggregate statistics of audience and critics ratings on the review site, rotten tomatoes[9]. The movie data file contains 10197 movies; some of which had no user ratings. We extracted the subset of $Ny = 10109$ movies that had been rated by the $Nx = 2113$ users. The resulting matrix density was 4%. The ratings are one of 10 distinct values ranging from 0.5 to 5.0 in increments of 0.5. As we show in Fig. 5, about half of the ratings are 4. The mean of the ratings is 3.4379 with a standard deviation of 1.0025.

We used the presence or absence of a tag as a global feature value. We also used the aggregate movie rating statistics provided, such as average critics rating and average community rating of each movie; a total of 13 values per movie. This set was repeated as appropriate to define global features used in the global linear bias model. The user-movie tag matrix was used to extract 100 inter-user and 100 inter-movie graph features. In addition, we also extracted 100 inter-movie interaction features using the movie-actor and movie-director graphs. Finally, we extracted 20 graph features from the movie-genre graph. We used a total of $P = 101$ user and $Q = 321$ movie kernels.

	Rank=5	rank=10
Global Model	0.9478 (0.0018)	-
Uniform Interaction	0.9471 (0.018)	.9481 (0.0025)
MF	0.7749 (0.0023)	.7706 (0.0016)
Weighted Interaction	**0.7742 (0.0024)**	**.7691 (0.001)**

Table 2: Average (std.) cross validation RMSE on Movielens

We report performance using the 5-fold validation. As shown in Table 6.2, the weighted interaction model outperformed matrix factorization. We found that the model assigned the over 96% of the user weight to the self-correlation kernel (Fig. 6), and less than 4% to the user-tag interactions. Similarly, 98% of the item kernel weight was assigned to the item self-correlation kernel (Fig. 7). Therefore, the resulting model was only slightly different from the matrix factorization result. There may be several reasons for this, the most likely reason is that the graphs available are not sufficiently (linearly) correlated with ratings values. In addition, the data is relatively dense, meaning that the information in the ratings history may be sufficient to explain most of the

[9]http://www.rottentomatoes.com

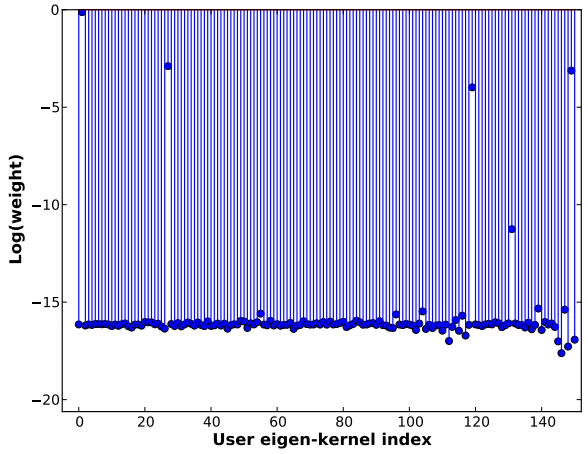

Figure 3: Last.fm user kernel log(weight). Identity(1%), User social network (93%), Tags (6%)

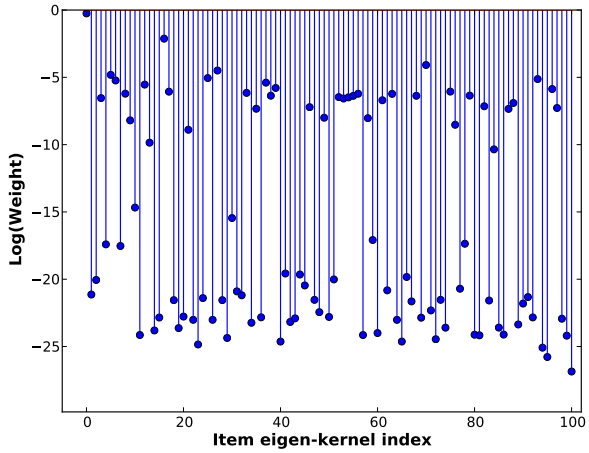

Figure 4: Last.fm item kernel log(weight). Identity(77%), Tags (23%)

observations. In either case, these results suggest that our model achieved our stated goal of automatically discarding un-correlated interactions.

We also tested the model on a time based split of the data as shown in Table 6.2. We sorted the ratings by time, then we extracted the first 90% of the ratings as training data, and predicted performance on the last 10%. We observed that though the performance of both the MF and the weighted interaction models degraded, learning the interaction still out-performed matrix factorization.

7. CONCLUSION AND FUTURE WORK

We have presented a recommender system that is tailored to the task of extracting information from pair-wise interaction networks. The results of our model indicate which interactions are most correlated with the recommendation. We showed that our model is robust and degenerates to the matrix factorization model when no relevant interactions are found.

We currently use standard techniques for learning the model parameters. One topic for future work is to investigate more efficient specialized approaches to solving the problem. We are also interested in additional validation on rich datasets with several sources of pair-wise interactions. The robustness of our model to cold start ratings can also be explored in more detail. In particular, we are interested in how the Nyström approximation for estimating the kernels values for new users and items will affect the performance of our model.

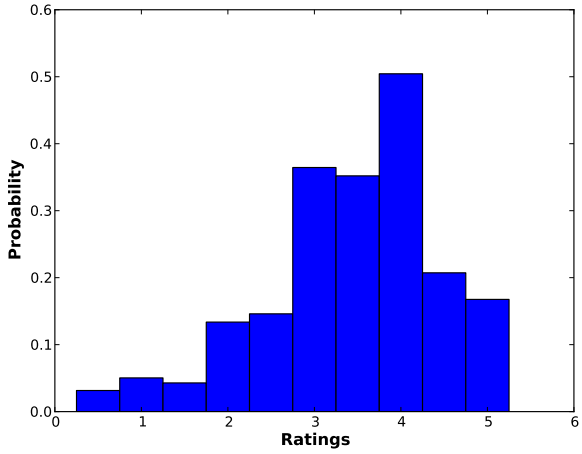

Figure 5: Movielens rating distribution $\mu = 3.4379,\ \sigma = 1.0025$.

	rank=5
Global Model	.9185
Uniform Interaction	.9184
MF	.8790
Weighted Interaction	**.8728**

Table 3: Global time split test RMSE on Movielens

8. ACKNOWLEDGMENT

This work was supported in part by NSF-IIS 1016614.

9. REFERENCES

[1] J. Abernethy, F. Bach, T. Evgeniou, and J.-P. Vert. Low-rank matrix factorization with attributes. *ArXiv Computer Science e-prints*, Nov. 2006.

[2] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6):734–749, 2005.

[3] A. Argyriou, M. Herbster, and M. Pontil. Combining graph laplacians for semi-supervised learning. In *Neural Information Processing Systems*, 2005.

[4] M. Belkin, P. Niyogi, and V. Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7:2399–2434, 2006.

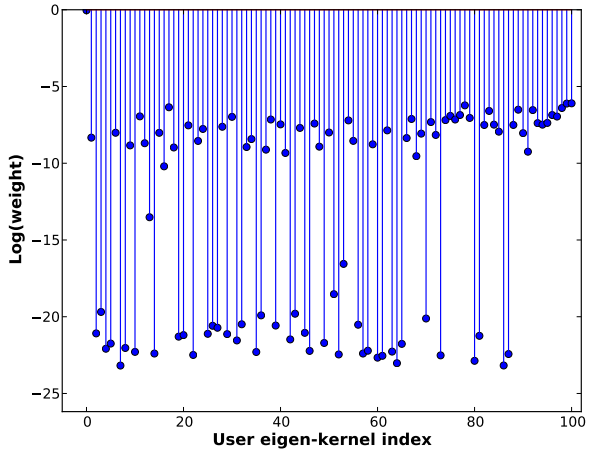

Figure 6: Movielens user kernel log(weight). Identity(96%), Tags (6%).

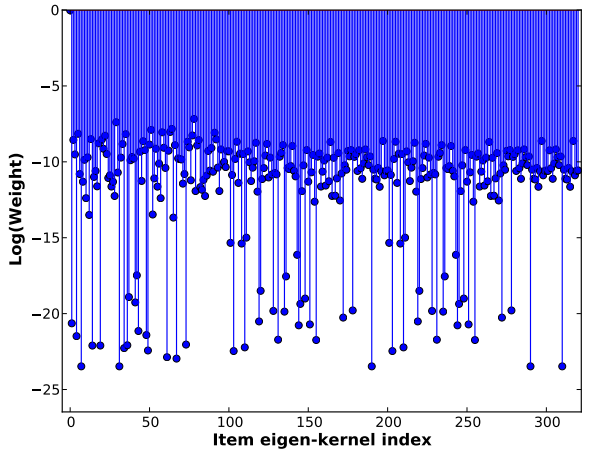

Figure 7: Movielens item kernel log(weight). Identity(98%), Tags (.8%), Actor (.3%), Director (.3%), Genre (.08%).

[5] Y. Bengio, J. Paiement, P. Vincent, O. Delalleau, N. Le Roux, and M. Ouimet. Out-of-sample extensions for LLE, Isomap, MDS, Eigenmaps, and Spectral Clustering. In S. Thrun, L. Saul, and B. SchÃűlkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, 2004.

[6] M. Deodhar and J. Ghosh. A framework for simultaneous co-clustering and learning from complex data. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 250–259, New York, NY, USA, 2007. ACM.

[7] J. A. Golbeck. *Computing and applying trust in web-based social networks*. PhD thesis, College Park, MD, USA, 2005. AAI3178583.

[8] M. Jamali and M. Ester. Trustwalker: a random walk model for combining trust-based and item-based recommendation. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 397–406, New York, NY, USA, 2009. ACM.

[9] M. Jamali and M. Ester. A matrix factorization technique with trust propagation for recommendation in social networks. In *Proceedings of the fourth ACM conference on Recommender systems*, RecSys '10, pages 135–142, New York, NY, USA, 2010. ACM.

[10] G. Jawaheer, M. Szomszor, and P. Kostkova. Comparison of implicit and explicit feedback from an online music recommendation service. In *Proceedings of the 1st International Workshop on Information Heterogeneity and Fusion in Recommender Systems*, HetRec '10, pages 47–51, New York, NY, USA, 2010. ACM.

[11] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42:30–37, 2009.

[12] U. Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17:395–416, December 2007.

[13] H. Ma, H. Yang, M. R. Lyu, and I. King. Sorec: social recommendation using probabilistic matrix factorization. In *Proceeding of the 17th ACM conference on Information and knowledge management*, CIKM '08, pages 931–940, New York, NY, USA, 2008. ACM.

[14] C. A. Micchelli and M. Pontil. Learning the kernel function via regularization. *J. Mach. Learn. Res.*, 6:1099–1125, December 2005.

[15] J. D. M. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 713–719, New York, NY, USA, 2005. ACM.

[16] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1257–1264. MIT Press, Cambridge, MA, 2008.

[17] B. Schölkopf, R. Herbrich, and A. J. Smola. A generalized representer theorem. In *Computational Learning Theory*, volume 2111 of *Lecture Notes in Computer Science*, pages 416–426. Springer Berlin / Heidelberg, 2001.

[18] K. Sinha and M. Belkin. Semi-supervised learning using sparse eigenfunction bases. In *Advances in Neural Information Processing Systems 22*. 2010.

[19] A. J. Smola and I. Kondor. Kernels and regularization on graphs. In B. Schölkopf and M. Warmuth, editors, *Proceedings of the Annual Conference on Computational Learning Theory*, Lecture Notes in Computer Science. Springer, 2003.