Kernelized Matrix Factorization for Collaborative Filtering

Xinyue Liu* Charu Aggarwal † Yu-Feng Li ‡ Xiangnan Kong* Xinyuan Sun*
Saket Sathe §

Abstract

Matrix factorization (MF) methods have shown great promise in collaborative filtering (CF). Conventional MF methods usually assume that the correlated data is distributed on a linear hyperplane, which is not always the case. Kernel methods are used widely in SVMs to classify linearly non-separable data, as well as in PCA to discover the non-linear embeddings of data. In this paper, we present a novel method to kernelize matrix factorization for collaborative filtering, which is equivalent to performing the low-rank matrix factorization in a possibly much higher dimensional space that is implicitly defined by the kernel function. Inspired by the success of multiple kernel learning (MKL) methods, we also explore the approach of learning multiple kernels from the rating matrix to further improve the accuracy of prediction. Since the right choice of kernel is usually unknown, our proposed multiple kernel matrix factorization method helps to select effective kernel functions from the candidates. Through extensive experiments on real-world datasets, we show that our proposed method captures the nonlinear correlations among data, which results in improved prediction accuracy compared to the state-of-art CF models.

1 Introduction

With the sheer growth of accessible online data, it becomes challenging as well as indispensable to develop technologies that helping people efficiently sift through the huge amount of information. *Collaborative filtering* is one of these technologies which has drawn much attention in recent years. It mainly serves as a recommender system [1] to automatically generate item recommendations based on past user-item feedbacks. Compared to other technologies in recommender systems, collaborative filtering does not need any content information about the items or users, it works by purely analyzing the preference patterns of users through the observed ratings. Besides, it also does not require any domain knowledge, which makes it easy to be applied on different datasets and systems. Recent advances in collaborative filtering have been paralleled by the success of matrix factorization technologies. The main idea of matrix factorization method is as follows. Suppose we are given a partially filled rating matrix, where each row stands for a user in the recommender system, and each column denotes an item in the system. The goal is to predict any missing entry (i, j) with the inner product of latent feature vectors for row (user) i and column (item) j as shown in Figure 1.

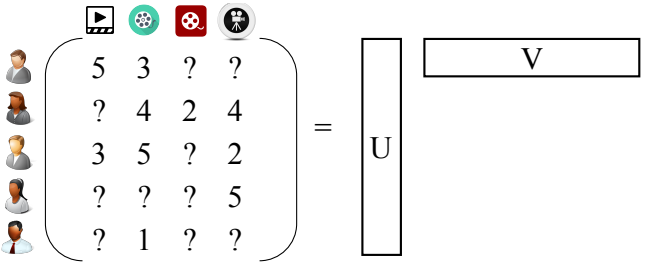


Figure 1: An illustration of using matrix factorization to predict the unobserved ratings. The rating matrix is factorized as the products of the two low-rank matrices of latent factors, denotes as **U** and **V**.

Matrix factorization methods have been extensively studied for collaborative filtering, and existing works [5, 7, 16] mainly focus on utilizing singular value decomposition (SVD) to derive the low-rank latent feature matrices. SVD is based on the assumption that the ratings are distributed on a linear hyperplane, thus they can be represented by the inner products of two low-rank structures. However, in many real-world collaborative filtering datasets, it is very hard or impossible to recover the full matrix well by solving low-rank factorizations linearly. In such cases, kernel methods can be helpful. Kernel methods work by embedding the data into a possibly higher dimensional feature space, in which the embeddings can be distributed on a linear hyperplane. Thus, it can be factorized into two feature matrices in that space. Although the embedding is implicitly de-

*Department of Computer Science, Worcester Polytechnic Institute
†IBM T. J. Watson Research Center
‡Nanjing University
§IBM Research Australia

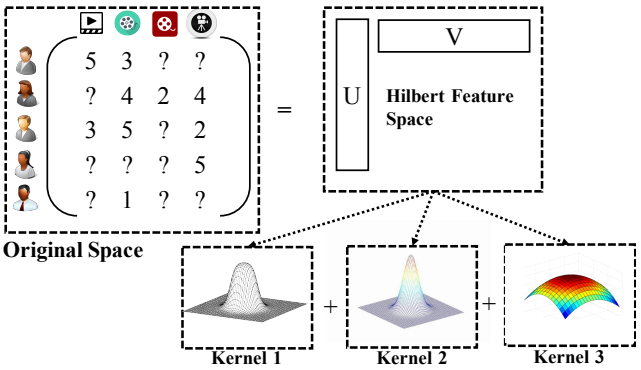


Figure 2: An illustration of kernelized low-rank matrix factorization for predicting the unobserved ratings. The two latent factor matrices $\mathbf{U}$ and $\mathbf{V}$ are embedded into a high-dimensional Hilbert feature space by a linear combination of kernel functions. The products of the two latent factor matrices reconstruct the rating matrix in original space nonlinearly.

fined, the inner products of the embeddings can be inferred by the kernel function, which is also called the kernel trick.

Despite its value and significance, few effort has been made on incorporating kernel methods into matrix factorization for collaborative filtering. And it is a much more challenging task due to the following reasons:

Lacking Side Information: Existing kernelized matrix factorization approaches [13, 27] usually require side information such as metadata, social graphs, text reviews *etc.* However, in the context of real-world recommender systems, such information is not always available, which limits the applicability of these approaches. Thus, without the help from side information of the users or items, applying kernel methods on pure matrix factorization is a more challenging task.

Latent Factors in Hilbert Feature Space: In conventional matrix factorization, the latent factors of users and items can be explicitly defined as two low-rank matrices. However, as to the kernelized matrix factorization problems, the matrices of latent factors exist in a Hilbert feature space, which is implicitly defined. To this end, applying alternative least square (ALS) or gradient descent approaches is hampered by the implication of the latent factors. Thus, it is challenging to devise effective kernel methods for the purpose of matrix completion.

Combining Multiple Kernels: A kernel function can capture a certain notion of similarity and efficiently embed data into a much higher dimensional space, but using only one single kernel function may lead to suboptimal performances. To address this problem, many methods have been proposed in the literature to learn linear combinations of multiple kernels, which are also an appealing strategy for improving kernelized matrix factorization. However, existing multiple kernel learning strategies are designed for classification models and regression models, such as SVM, ridge regression, *etc.* It is challenging to combine multiple kernel learning and matrix factorization, especially when considering the large scale of collaborative filtering tasks compared to conventional classification or regression problems.

In order to solve the above issues, we propose a novel solution called MKMF, combining matrix factorization with kernel methods and multiple kernel learning. Different from previous work, the proposed MKMF can capture the non-linear relationships in the rating data, without requiring any side information. Empirical studies on real-world datasets show that the proposed MKMF can improve the accuracy of prediction for collaborative filtering.

The rest of the paper is organized as follows. We start by introducing the preliminary concepts and giving the problem formulation in Section 2. Then we present our proposed MKMF approach in section 3. Section 4 reports the experiment results. We also briefly review on related works of collaborative filtering in Section 5. In section 6, we conclude the paper.

2 Problem Formulation

2.1 Notations In this section, we briefly describe the general problem setting of collaborative filtering with some preliminaries of our proposed method.

Suppose a set of m users and a set of n items are given, each observed rating is a tuple (u, i, r_{ui}) denoting the rating assigned to item i by user u, where the user index $u \in \{1, \ldots, m\}$, the item index $i \in \{1, \ldots, n\}$, and the rating values $r_{ui} \in \mathbb{R}$. We also assume that each user can only rate an item once, so all such ratings can be arranged into an $m \times n$ matrix $\mathbf{R}$ whose ui-th entry equals r_{ui}. If user p has not rated item q, then the corresponding entry r_{pq} in $\mathbf{R}$ is unknown or unobserved. We define the observed entries indexed by the set $\Omega = \{(u, i) : r_{ui} \text{ is observed}\}$, and $|\Omega|$ the number of observed ratings. Due to the well-known sparsity problem, typically $|\Omega|$ is much smaller than $m \times n$.

We use Ω_u to denote the indices of observed ratings of the u-th row $\mathbf{r}_u$, and Ω^i the indices of observed ratings of the i-th column $\mathbf{r}_{:,i}$. For examples, if $\mathbf{r}_u$'s ratings are all unobserved except the 1st and the 3rd values, then:

- $\Omega_u = (1, 3)^\top$ and $\mathbf{r}_{\Omega_u} = (r_{u1}, r_{u3})^\top$.
- Given a matrix $\mathbf{A}$, $\mathbf{A}_{:,\Omega_u}$ denotes a sub matrix formed by the 1st column and 3rd column of $\mathbf{A}$.

Similarly, if $\mathbf{r}_u$'s ratings are all unobserved except

Table 1: Important Notations.

Symbol	Definition
$\mathbf{R}$ and $\hat{\mathbf{R}}$	The partially observed rating matrix and the inferred dense rating matrix
$\mathbf{U}$ and $\mathbf{V}$	The user latent factors matrix and the item latent factors matrix
$\Omega = \{(u,i)\}$	the index set of observed ratings, for $(u,i) \in \Omega$, the r_{ui} is observed in $\mathbf{R}$
Ω_u	indices of observed ratings of the u-th row of $\mathbf{R}$
Ω^i	indices of observed ratings of the u-th column of $\mathbf{R}$
$\mathbf{A}_{:,\Omega_u}$ or $\mathbf{A}_{:,\Omega^i}$	Sub matrix of $\mathbf{A}$ formed by the columns indexed by Ω_u or Ω^i
$\{\mathbf{d}_1, \ldots, \mathbf{d}_k\}$	the set of dictionary vectors
$\phi(\cdot)$	the implicit mapping function of some kernel
$\mathbf{\Phi} = (\phi(\mathbf{d}_1), \ldots, \phi(\mathbf{d}_k))$	the matrix of embedded dictionary in Hilbert feature space
$\mathbf{a}_u$ and $\mathbf{b}_i$	the dictionary weight vector for user u and the dictionary weight vector for item i
$\mathbf{A}$ and $\mathbf{B}$	the dictionary weight matrix of users and the dictionary weight matrix of items.
$\{\kappa_1, \ldots, \kappa_p\}$	the set of base kernel functions
$\{\mathbf{K}_1, \ldots, \mathbf{K}_p\}$	the set of base kernel matrices

the 2nd and the 4th values, then:

- $\Omega^i = (2,4)^\top$ and $\mathbf{r}_{:,\Omega^i} = (r_{2i}, r_{4i})^\top$.
- Given a matrix $\mathbf{A}$, $\mathbf{A}_{:,\Omega^i}$ denotes a sub matrix formed by the 2nd column and 4th column of $\mathbf{A}$.

The goal of collaborative filtering is to estimate the unobserved ratings $\{r_{ui} \mid (u,i) \notin \Omega\}$ based on the observed ratings.

2.2 Matrix Factorization Matrix factorization is widely used to solve matrix completion problem like collaborative filtering as we defined above. The idea of matrix factorization is to approximate the observed matrix $\mathbf{R}$ as the product of two low-rank matrices:

$$\mathbf{R} \approx \mathbf{U}^\top \mathbf{V},$$

where $\mathbf{U}$ is a $k \times m$ matrix and $\mathbf{V}$ is a $k \times n$ matrix. The parameter k controls the rank of the factorization, which also denotes the number of latent features for each user and item. Note that in most cases, we have $k \ll \min(m,n)$.

Once we obtain the low-rank decomposition matrices $\mathbf{U}$ and $\mathbf{V}$, each prediction for rating assigned to item i by user u can be made by:

$$\hat{r}_{ui} = \sum_{j=1}^{k} u_{ju} v_{ji} = \mathbf{u}_u^\top \mathbf{v}_i, \tag{2.1}$$

where u_{ju} denotes the element in j-th row and u-th column of matrix $\mathbf{U}$, v_{ji} denotes the element in j-th row and i-th column of matrix $\mathbf{V}$, $\mathbf{u}_u$ denotes the u-th column of $\mathbf{U}$, and $\mathbf{v}_i$ denotes the i-th column of $\mathbf{V}$.

The low-rank parameter matrices $\mathbf{U}$ and $\mathbf{V}$ can be found by solving the following problem:

$$\underset{\mathbf{U},\mathbf{V}}{\text{minimize}} \; ||P_\Omega(\mathbf{R} - \mathbf{U}^\top \mathbf{V})||_F^2 + \lambda(||\mathbf{U}||_F^2 + ||\mathbf{V}||_F^2) \tag{2.2}$$

where the projection $P_\Omega(\mathbf{X})$ is the matrix with observed elements of $\mathbf{X}$ preserved, and the unobserved entries replaced with 0, $||\cdot||_F^2$ denotes the Frobenius 2-norm and λ is the regularization term for avoiding over-fitting. This problem is not convex in terms of $\mathbf{U}$ and $\mathbf{V}$, but it is bi-convex. Thus, alternating minimization algorithms such as ALS [8, 10] can be used to solve equation (2.2). Suppose $\mathbf{U}$ is fixed, and we target to solve (2.2) for $\mathbf{V}$. Then we can decompose the problem into n separate ridge regression problems, for the j-th column of $\mathbf{V}$, the ridge regression can be formalized as:

$$\underset{\mathbf{v}_j}{\text{minimize}} \; ||\mathbf{r}_{:,\Omega^j} - \mathbf{U}_{:,\Omega^j}^\top \mathbf{v}_j||_F^2 + \lambda ||\mathbf{v}_j||_2 \tag{2.3}$$

where $\mathbf{r}_{:,\Omega^j}$ denotes the j-th column of $\mathbf{R}$ with unobserved element removed, the corresponding columns of $\mathbf{U}$ are also removed to derive $\mathbf{U}_{:,\Omega^j}$, as we defined in Section 2.1. The closed form solution for above ridge regression is given by:

$$\hat{\mathbf{v}}_j = (\mathbf{U}_{:,\Omega^j} \mathbf{U}_{:,\Omega^j}^\top + \lambda \mathbf{I})^{-1} \mathbf{U}_{:,\Omega^j} \mathbf{r}_{:,\Omega^j}. \tag{2.4}$$

Since each of the n separate ridge regression problems leads to a solution of $\hat{\mathbf{v}} \in \mathbb{R}^k$, stacking these n separate $\hat{\mathbf{v}}$'s together gives a $k \times n$ matrix $\hat{\mathbf{V}}$. Symmetrically, with $\mathbf{V}$ fixed, we can find a solution of $\hat{\mathbf{U}}$ by solving m separate ridge regression problems. Repeat this procedure until convergence eventually leads to the solution of $\hat{\mathbf{U}}$ and $\hat{\mathbf{V}}$.

3 Multiple Kernel Collaborative Filtering

The matrix factorization method assumes the data of matrix $\mathbf{R}$ is distributed on a linear hyperplane, which is not always the case. Kernel methods [11, 21] can be helpful when data of matrix $\mathbf{R}$ is distributed on a nonlinear hyperplane.

3.1 Kernels Kernel methods work by embedding the data into a high-dimensional (possibly infinite-dimensional) feature space $\mathcal{H}$, where the embedding is implicitly specified by a kernel. Suppose we have a kernel whose corresponding feature space mapping is defined as $\phi : \mathcal{X} \to \mathcal{H}$, where $\mathcal{X}$ is the original space and $\mathcal{H}$ is the Hilbert feature space. Given a data vector $\mathbf{x} \in \mathcal{X}$, then the embedding of $\mathbf{x}$ in $\mathcal{H}$ can be denoted as $\phi(\mathbf{x})$. Although $\phi(\mathbf{x})$ is implicit here, the inner product of data point in the feature space is explicit and can be derived as $\phi(\mathbf{x})^\top \phi(\mathbf{x}') = \kappa(x, x') \in \mathbb{R}$, where κ is the so-called kernel function of the corresponding kernel. A popular kernel function that has been widely used is the Gaussian kernel (or RBF kernel):

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\sigma^2}\right), \tag{3.5}$$

where σ^2 is known as the *bandwidth* parameter. Different kernel functions specify different embeddings of the data and thus can be viewed as capturing different notions of correlations.

3.2 Dictionary-based Single Kernel Matrix Factorization Suppose we have k dictionary vectors $\{\mathbf{d}_1, \ldots, \mathbf{d}_k\}$, where $\mathbf{d} \in \mathbb{R}^d$. Then we assume that the feature vector $\phi(\mathbf{u}_u)$ associated to $\mathbf{u}_u$ can be represented as a linear combination of the dictionary vectors in kernel space as follows:

$$\phi(\mathbf{u}_u) = \sum_{j=1}^{k} a_{uj}\phi(\mathbf{d}_j) = \mathbf{\Phi}\mathbf{a}_u, \tag{3.6}$$

where $a_{uj} \in \mathbb{R}$ denotes the weights of each dictionary vector, $\phi(\mathbf{d}_i)$ denotes the feature vector of $\mathbf{d}_i$ in Hilbert feature space, $\mathbf{a}_u = (a_{u1}, \ldots, a_{uk})^\top$ and $\mathbf{\Phi} = (\phi(\mathbf{d}_1), \ldots, \phi(\mathbf{d}_k))$. Similarly we also assume that the feature vector $\phi(\mathbf{v}_i)$ associated to $\mathbf{v}_i$ can be represented as:

$$\phi(\mathbf{v}_i) = \sum_{j=1}^{k} b_{ij}\phi(\mathbf{d}_j) = \mathbf{\Phi}\mathbf{b}_i, \tag{3.7}$$

where b_ij is the weight for each dictionary vector and $\mathbf{b}_i = (b_{i1}, \ldots, b_{ik})^\top$. Thus for each user $u \in \{1, \ldots, m\}$ we have a weight vector $\mathbf{a}_u$, for each item $i \in \{1, \ldots, n\}$, we have a weight vector $\mathbf{b}_i$.

Consider the analog of (2.3), when all $\phi(\mathbf{u}_u)$ are fixed, *i.e.* the weight matrix $\mathbf{A} = (\mathbf{a}_1, \ldots, \mathbf{a}_m)$ is fixed, and we wish to solve for all $\phi(\mathbf{v}_i)$, *i.e.* to solve for the weight matrix $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$

$$\underset{\phi(\mathbf{v}_i)\in\mathcal{H}}{\text{minimize}} \sum_u (r_{ui} - \phi(\mathbf{u}_u)^\top \phi(\mathbf{v}_i))^2 + \lambda\phi(\mathbf{v}_i)^\top\phi(\mathbf{v}_i) \tag{3.8}$$

It is easy to see

$$\phi(\mathbf{u}_u)^\top \phi(\mathbf{v}_i) = \mathbf{a}_u^\top \mathbf{\Phi}^\top \mathbf{\Phi}\mathbf{b}_i = \mathbf{a}_u^\top \mathbf{K}\mathbf{b}_i, \tag{3.9}$$

$$\phi(\mathbf{v}_i)^\top \phi(\mathbf{v}_i) = \mathbf{b}_i^\top \mathbf{\Phi}^\top \mathbf{\Phi}\mathbf{b}_i = \mathbf{b}_i^\top \mathbf{K}\mathbf{b}_i, \tag{3.10}$$

where $\mathbf{K} = \mathbf{\Phi}^\top\mathbf{\Phi}$ is the Gram matrix (or kernel matrix) of the set of dictionary vectors $\{\mathbf{d}_1, \ldots, \mathbf{d}_k\}$. So we can rewrite (3.8) as

$$\underset{\mathbf{b}_i\in\mathbb{R}^k}{\text{minimize}} \sum_u (r_{ui} - \mathbf{a}_u^\top \mathbf{K}\mathbf{b}_i)^2 + \lambda\mathbf{b}_i^\top\mathbf{K}\mathbf{b}_i \tag{3.11}$$

which is equivalent to

$$\underset{\mathbf{b}_i\in\mathbb{R}^k}{\text{minimize}} ||\mathbf{r}_{:,\Omega^i} - \mathbf{A}_{:,\Omega^i}^\top \mathbf{K}\mathbf{b}_i||_F^2 + \lambda\mathbf{b}_i^\top\mathbf{K}\mathbf{b}_i, \tag{3.12}$$

(3.12) is similar to kernel ridge regression, the closed form solution is given by

$$\hat{\mathbf{b}}_i = (\mathbf{K}^\top \mathbf{A}_{:,\Omega^i}\mathbf{A}_{:,\Omega^i}^\top \mathbf{K} + \lambda\mathbf{K})^\dagger \mathbf{K}^\top \mathbf{A}_{:,\Omega^i}\mathbf{r}_{:,\Omega^i} \tag{3.13}$$

Stacking n separate $\hat{\mathbf{b}}$ together, we get the estimated $\hat{\mathbf{B}} = (\hat{\mathbf{b}}_1, \ldots, \hat{\mathbf{b}}_n)$, which is a $k \times n$ matrix. Symmetrically, with $\mathbf{B}$ fixed, we can find a solution of $\hat{\mathbf{A}}$ by solving m separate optimization problem like (3.12). In this case, the closed form solution for each $\hat{\mathbf{a}}_u$ is given by:

$$\hat{\mathbf{a}}_u = (\mathbf{K}^\top \mathbf{B}_{:,\Omega_u}\mathbf{B}_{:,\Omega_u}^\top \mathbf{K} + \lambda\mathbf{K})^\dagger \mathbf{K}^\top \mathbf{B}_{:,\Omega_u}\mathbf{r}_{\Omega_u} \tag{3.14}$$

Algorithm 1 Multiple Kernel Matrix Factorization

Require: $k, d, \{\kappa_1, \ldots, \kappa_p\}, \mathbf{R}, \Omega, \lambda, iter_{max}$
allocate $\mathbf{A} \in \mathbb{R}^{k\times m}, \mathbf{B} \in \mathbb{R}^{k\times n}, \boldsymbol{\mu} \in \mathbb{R}^p, \mathcal{D} = \{\mathbf{d}_i \in \mathbb{R}^d : 1 \le i \le k\}, \{\mathbf{K}_i \in \mathbb{R}^{k\times k} : 1 \le i \le p\}, \mathbf{K} \in \mathbb{R}^{k\times k}$
initialize $\boldsymbol{\mu} = (\frac{1}{p}, \ldots, \frac{1}{p})^\top$, $\mathbf{A}$, $\mathbf{B}$, $\mathcal{D}$
for $i \leftarrow 1, p$ **do**
    $\mathbf{K}_i \leftarrow (\kappa_i(\mathbf{d}_h, \mathbf{d}_j))_{1\le h,j\le k}$
end for
$iter \leftarrow 0$
repeat
    $\mathbf{K} \leftarrow \sum_{i=1}^p \mu_i \mathbf{K}_i$
    Update $\mathbf{B}$ as shown in (3.13)
    Update $\mathbf{A}$ as shown in (3.14)
    Update $\boldsymbol{\mu}$ by solving (3.16)
until $iter = iter_{max}$ or convergence
Return $\mathbf{A}, \mathbf{B}, \boldsymbol{\mu}, \mathbf{K}$

3.3 Multiple Kernel Matrix Factorization Multiple kernel learning(MKL) [2, 6, 12] has been widely used on improving the performance of classification and regression tasks, the basic idea of MKL is combine multiple kernels instead of using a single one.

Formally, suppose we have a set of p positive defined base kernels $\{\mathbf{K}_1, \ldots, \mathbf{K}_p\}$, then we aim to learn a kernel based prediction model by identifying the best linear combination of the p kernels, that is, a weighted combinations $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_p)^\top$. The learning task can be cast into following optimization:

$$\begin{aligned}\underset{\mathbf{a}_u, \mathbf{b}_i \in \mathbb{R}^k}{\text{minimize}} \quad & \sum_{(u,i)\in\Omega} (r_{ui} - \underbrace{\sum_{j=1}^{p} \mu_j \mathbf{a}_u^\top \mathbf{K}_j \mathbf{b}_i}_{\boldsymbol{\upsilon}_{ui}^\top \boldsymbol{\mu}})^2 + \\ & \underbrace{\lambda(\mathbf{b}_i^\top \sum_{j=1}^{p} \mu_j \mathbf{K}_j \mathbf{b}_i + \mathbf{a}_u^\top \sum_{j=1}^{p} \mu_j \mathbf{K}_j \mathbf{a}_u)}_{\boldsymbol{\gamma}_{ui}^\top \boldsymbol{\mu}}\end{aligned} \tag{3.15}$$

where $\boldsymbol{\mu} \in \mathbb{R}_+^p$ and $\boldsymbol{\mu}^\top \mathbf{1}_p = 1$. It is convenient to introduce the vectors $\boldsymbol{\upsilon}_{ui} = (\mathbf{a}_u^\top \mathbf{K}_1 \mathbf{b}_i, \ldots, \mathbf{a}_u^\top \mathbf{K}_p \mathbf{b}_i)^\top$, $\boldsymbol{\gamma}_{ui} = (\mathbf{a}_u^\top \mathbf{K}_1 \mathbf{a}_u + \mathbf{b}_i^\top \mathbf{K}_1 \mathbf{b}_i, \ldots, \mathbf{a}_u^\top \mathbf{K}_p \mathbf{a}_u + \mathbf{b}_i^\top \mathbf{K}_p \mathbf{b}_i)^\top$.

Rearrange optimization (3.15),

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{\mu}^\top \mathbf{Y} \boldsymbol{\mu} + \mathbf{Z} \boldsymbol{\mu} \\ \text{subject to} \quad & \boldsymbol{\mu} \succeq 0 \\ & \mathbf{1}_p^\top \boldsymbol{\mu} = 1 \end{aligned} \tag{3.16}$$

where $\mathbf{Y} = \sum_{(u,i)\in\Omega} \boldsymbol{\upsilon}_{ui} \boldsymbol{\upsilon}_{ui}^\top$ and $\mathbf{Z} = \sum_{(u,i)\in\Omega} (\lambda - 2r_{ui}) \boldsymbol{\gamma}_{ui}$. When all $\boldsymbol{\upsilon}_{ui}$ and $\boldsymbol{\gamma}_{ui}$ are fixed, *i.e.* $\mathbf{A}$ and $\mathbf{B}$ are fixed, the optimization problem (3.16) is known as a *quadratic programming*, which can be solved efficiently by software package like CVXOPT[1] or CVX[2].

Now we can put all optimization problems together to build up our multiple kernel matrix factorization algorithm, which is summarized in algorithm 1. Initially, given rank value k, the dictionary dimension d and p base kernel functions $\{\kappa_1, \ldots, \kappa_p\}$, the algorithm randomly initializes k dictionary vectors $\mathcal{D} = (\mathbf{d}_1, \ldots, \mathbf{d}_k)$, then computes the p base kernel matrices as $\mathbf{K}_i = (\kappa_i(\mathbf{d}_h, \mathbf{d}_j))_{1 \le h,j \le k}$. The algorithm also initializes the kernel weight vector as $\boldsymbol{\mu}^0 = (\frac{1}{p}, \ldots, \frac{1}{p})^\top$, and generates low-rank matrix $\mathbf{A}^0$ and $\mathbf{B}^0$ randomly. So initially, the compound kernel matrix $\mathbf{K}^0 = \sum_{1 \le i \le p} \mu_i^0 \mathbf{K}_i$. After obtaining all above instantiations, we first let $\mathbf{A}^0$ and $\boldsymbol{\mu}^0$ fixed, find $\mathbf{B}^1$ by solving m separate optimization problems like (3.12), each solution can be obtained directly by computing the closed form expression shown in 3.13. Similarly, we then let $\mathbf{B}^1$ and $\boldsymbol{\mu}^0$ fixed, following the same approach to get $\mathbf{A}^1$. At last, $\mathbf{A}^1$ and $\mathbf{B}^1$ are fixed, we can obtain $\boldsymbol{\mu}^1$ by solving 3.16 using convex optimization package mentioned before. Repeat this ALS-like procedure until the algorithm converges or reaches the predefined maximum number of iterations. We then define optimal solutions obtained by above iterative procedure are $\overset{*}{\mathbf{A}}$,$\overset{*}{\mathbf{B}}$ and $\overset{*}{\boldsymbol{\mu}}$, the corresponding compound kernel matrix is denoted as $\overset{*}{\mathbf{K}} = \sum_{i=1}^{p} \overset{*}{\mu}_i \mathbf{K}_i$. Thus, for each test tuple (u, i, r_{ui}), the prediction made by our algorithm is $\hat{r}_{ui} = \mathbf{a}_u^\top \overset{*}{\mathbf{K}} \mathbf{b}_i$, which also is the element in the u-th row and i-th column of the recovered matrix $\hat{\mathbf{R}} = \overset{*}{\mathbf{A}}^\top \overset{*}{\mathbf{K}} \overset{*}{\mathbf{B}}$. The rating inference is also summarized in algorithm 2.

Algorithm 2 Rating Inference

Require: $\hat{\Omega}, \overset{*}{\mathbf{K}}, \overset{*}{\mathbf{A}}, \overset{*}{\mathbf{B}}$
1: allocate $\mathcal{P}$
2: **for** $u, i \in \hat{\Omega}$ **do**
3: $\quad \hat{r}_{ui} \leftarrow \overset{*}{\mathbf{a}}_u^\top \overset{*}{\mathbf{K}} \mathbf{b}_i$
4: $\quad$ add $\hat{r}_{ui}$ to $\mathcal{P}$
5: **end for**
6: **Return** $\mathcal{P}$.

4 Experiments

To examine the performance of in addressing the collaborative filtering, we conducted extensive experiments on several real-world datasets. In this section, we introduce the datasets we used and the experiments we performed respectively, then we present the experimental results as well as the analysis.

4.1 Datasets We evaluated the proposed method on 6 real world datasets of recommender system: `MovieLens`, `Jester`, `Flixster`, `Dating Agency`, `Yahoo Music` and `ASSISTments`. The details are summarized in Table 2. Note that the scale of rating in each dataset is different. For instance, the ratings in Jester dataset are continuous values ranges from -10 to 10, while the Yahoo Music dataset contains 100 rating classes from 1 to 100. For each dataset, we sampled a subset with 1,000 users and 1,000 items. The 1000 users are selected randomly while the 1000 items selected are most frequently rated items[3].

4.2 Compared Methods In order to demonstrate the effectiveness of our multiple kernel matrix factorization method, we compared the proposed framework with following existing baselines.

- AVG: First we implemented a naive baseline called AVG, which predicts the unobserved rating assign to item i by user u as $\hat{r}_{ui} = \alpha_i + \beta_u$, where α_i is

[1] http://cvxopt.org
[2] http://cvxr.com/cvx/

[3] Jester dataset only contains 100 items.

Table 2: Summary of Datasets

Dataset	# of users	# of items	Density (%)
MovieLens	6,040	3,900	6.3
Jester	73,421	100	55.8
Flixster	147,612	48,784	0.11
Dating Agency	135,359	168,791	0.76
Yahoo Music	1,948,882	98,211	0.006
ASSISTments	46,627	179,084	0.073

Table 3: Summary of compared methods.

Method	**Type**	**Kernelized**	**Publication**
AVG	Memory-Based	No	[14]
IVC-COS	Memory-Based	No	[20]
IVC-PERSON	Memory-Based	No	[20]
SVD	Model-Based	No	[5]
MF	Model-Based	No	[10]
KMF	Model-Based	Single Kernel	This paper
MKMF	Model-Based	Multiple Kernels	This paper

the mean score of item i in training data, and β_u is the average bias of user u computed as

$$\beta_u = \frac{\sum_{(u,i)\in\Omega} r_{ui} - \alpha_i}{||\{(u,i)\in\Omega\}||}, \text{where } \alpha_i = \frac{\sum_{(u,i)\in\Omega} r_{ui}}{||\{(u,i)\in\Omega\}||}$$

- IVS-COS (Item-based + cosine vector similarly): For memory based methods we implemented IVS-COS. It predicts the rating assigning to item i by user u by searching for the neighboring set of items rated by user u using cosine vector similarity [20].
- IVS-PEARSON (Item-based + Pearson vector similarly): Another memory based method we compare with is the item based model which uses Pearson correlation [20]. It is similar to IVS-COS except using different metric of similarity.
- SVD: We also implemented a model based method called Funk-SVD [5]. It is similar to the matrix factorization approach we discussed in Section 2.2, but it is implemented by using gradient descent instead of alternative least square.
- MF: MF (Matrix Factorization) is a model-based CF method that discussed in Section 2.2.
- KMF: We also compare the proposed MKMF with its single-kernel version KMF which only employs one kernel function at a time, the details are discussed in Section 3.2.
- MKMF: Another proposed method for kernelized collaborative filtering. The only difference between MKMF and KMF is that MKMF combine multiple kernel functions while KMF only use one kernel function. The details of MKMF are discussed in Section 3.2.

For a fair comparison, the maximum number of iterations in the methods SVD, MF, KMF and MKMF are all fixed as 20.

4.3 Evaluation Metric Collaborative filtering algorithms are evaluated by the accuracy of their predict ratings. One commonly used performance metric for rating accuracy is the root mean square error (RMSE):

$$\text{RMSE} = \sqrt{\frac{\sum_{(u,i)\in\Omega} \left(r_{ui} - \hat{r}_{ui}\right)^2}{|\Omega|}} \tag{4.17}$$

4.4 Evaluation Protocol For each dataset, we select the 1,000 most frequently rated items and randomly draw 1,000 users to generate a matrix with the dimension of 1000×1000 (1000×100 for Jester). Two experimental settings are tested in this paper respectively. In the first setting, we randomly select one rating from each user for test and the remaining ratings for training. The random selection is repeated 5 times independently for each dataset. In the second setting, we randomly select 3 ratings from each user for test and the remaining ratings for training. The random selection is also repeated 5 times independently for each dataset. We denote the first setting as `Leave 1`, and the second setting as `Leave 3`, which has sparser training matrix than the setting of `Leave 1`. The average performances with the rank of each method are reported.

4.5 Results and Discussion Following the setting of [12], we first use a small base kernel set of three simple kernels: a linear kernel function $\kappa_1(\mathbf{x},\mathbf{x}') = \mathbf{x}^\top\mathbf{x}$, a 2-degree polynomial kernel function $\kappa_2(\mathbf{x},\mathbf{x}') = (1+\mathbf{x}^\top\mathbf{x})^2$, and a Gaussian kernel (rbf kernel) function $\kappa_3(\mathbf{x},\mathbf{x}') = \exp(-0.5(\mathbf{x}-\mathbf{x}')^\top(\mathbf{x}-\mathbf{x}')/\sigma)$ with $\sigma = 0.5$. We implement the proposed MKMF method to learn a linear combination of the three base kernels by solving the optimization problem (3.15). The proposed KMF method is also tested using the three base kernels respectively.

Empirical results on the six real-word datasets are summarized in Table 4 and 5. Based on the results, we have made following interesting observations:

- The proposed MKMF method generally outperforms the other baselines on almost all datasets in terms of RMSE. On the ASSISTments dataset, AVG achieves best RMSE in two settings among all compared methods, possibly because the ratings are binary in ASSITments. However, our proposed

Table 4: Results (RMSE(rank)) of `Leave 1` on the real-world datasets.

Methods	Dataset												Ave. Rank
	Flixster		Yahoo Music		MovieLens		Jester		Dating		ASSISTments		
MKMF	0.8270	(1)	18.3036	(1)	0.8223	(1)	4.0398	(1)	1.6697	(1)	0.7920	(2)	1.1667
KMF (LINEAR)	0.8290	(4)	18.3734	(3)	0.8241	(3.5)	4.0471	(2.5)	1.6804	(4)	0.7933	(4)	3.5
KMF (POLY)	0.8289	(3)	18.3766	(4)	0.8241	(3.5)	4.0472	(4)	1.6797	(3)	0.7934	(5)	3.75
KMF (RBF)	0.8292	(5)	18.3883	(5)	0.8246	(5)	4.0471	(2.5)	1.6701	(2)	0.7927	(3)	3.75
MF	0.8286	(2)	18.3214	(2)	0.8235	(2)	4.0549	(5)	1.6849	(5)	0.8001	(6)	3.6667
SVD	0.9441	(9)	26.2255	(9)	0.9406	(7)	4.2794	(6)	1.7920	(7)	0.8919	(9)	7.8333
IVC-COS	0.9223	(7)	22.9477	(7)	1.0016	(8)	4.6137	(8)	2.7052	(8)	0.8106	(7)	7.5
IVC-PEARSON	0.9226	(8)	22.9486	(8)	1.0020	(9)	4.6142	(9)	2.8209	(9)	0.8446	(8)	8.5
AVG	0.9006	(6)	22.4159	(6)	0.8887	(6)	4.3867	(7)	1.7349	(6)	0.7658	(1)	5.3333

Table 5: Results (RMSE (rank)) of `Leave 3` on the real-word datasets.

Methods	Dataset												Ave. Rank
	Flixster		Yahoo Music		MovieLens		Jester		Dating		ASSISTments		
MKMF	0.8153	(1)	18.5034	(1)	0.8168	(1)	4.0809	(1)	1.6675	(5)	0.7917	(3.5)	2.0833
KMF (LINEAR)	0.8163	(4.5)	18.5426	(3)	0.8178	(2.5)	4.0826	(3)	1.6561	(2)	0.7917	(3.5)	3.0833
KMF (POLY)	0.8163	(4.5)	18.5500	(4)	0.8179	(4)	4.0824	(2)	1.6562	(3)	0.7919	(5)	3.75
KMF (RBF)	0.8157	(3)	18.5521	(5)	0.8186	(5)	4.0829	(4)	1.6488	(1)	0.7903	(2)	3.3333
MF	0.8155	(2)	18.5375	(2)	0.8178	(2.5)	4.0865	(5)	1.6663	(4)	0.7955	(6)	3.5833
SVD	0.9335	(9)	24.3480	(9)	0.9346	(7)	4.2601	(6)	1.8226	(7)	0.8852	(9)	7.8333
IVC-COS	0.9050	(7)	23.0763	(7)	1.0034	(8)	4.6282	(8)	2.6964	(8)	0.8069	(7)	7.5
IVC-PEARSON	0.9051	(8)	23.0767	(8)	1.0047	(9)	4.6284	(9)	2.8212	(9)	0.8464	(8)	8.5
AVG	0.8868	(6)	22.6101	(6)	0.8865	(6)	4.3656	(7)	1.7561	(6)	0.7632	(1)	5.3333

MKMF still outperforms all other methods in `Leave 1` and the proposed KMF with RBF kernel outperforms all other methods in `Leave 3`. It shows incorporating kernels into matrix factorization helps the model capturing the non-linear correlation among the data, and the overall accuracy is improved.

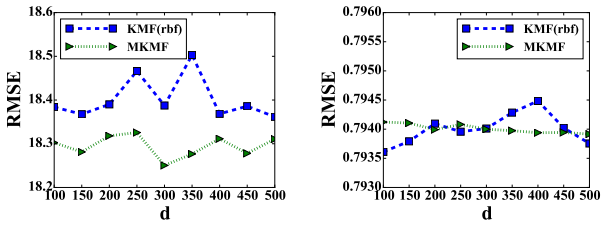
- It is interesting to note that the linear kernel KMF outperforms MF on 3 out of 6 datasets. The only difference between linear kernel KMF and MF is that linear kernel KMF employs a set of dictionary vectors to embed the data into a different space of the same dimension.

- In some cases, KMFs are outperformed by MF, but the MKMF that combining three kernels used by KMFs outperforms MF and other baselines. It proves that by learning the weight of multiple kernels based on the rating data, MKMF sometimes find better embeddings that are hard for KMF to find.

- The performance of MKMF is slightly worse in `Leave 3` than in `Leave 1` while KMF keeps unaffected. It indicates that the multiple kernel learning algorithm is more sensitive to the sparsity problem than solving kernel ridge regression.

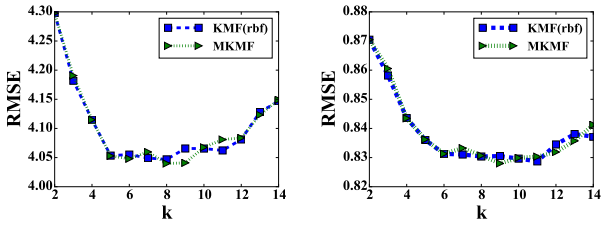
4.6 Parameter Studies Figure 3(a) and Figure 3(b) shows the kernelized methods' sensitivity of parameter d, which stands for the dimensions of the dictionary vectors. Generally, according to the figures, both KMF and MKMF are not very sensitive to the value of d once we choose relatively large d. For Yahoo Music dataset, the optimal choice of d is around 300, as to ASSISTments, the preferred $d \approx 100$. The situations of other datasets are similar, so they are omitted here due to the page limitation. Intuitively, the larger d we choose, the more information can be captured by the dictionary. However, since in both KMF and MKMF, the dictionary vectors are eventually embedded into a Hilbert feature space with much higher dimension than d. It is reasonable to observe the insensitive pattern is shown in Figure 3(a) and 3(b).

Besides, we also study the performances of our proposed methods upon different values of k, which denotes the rank of the two low-rank feature matrices. It is well-known the parameter k should be tuned via cross-validation or multiple rounds of training/test split



(a) Yahoo Music Dataset (b) ASSISTments Dataset

Figure 3: Comparison of different values of d (dimension of dictionary vectors)



(a) Jester Dataset (b) Flixster Dataset

Figure 4: Comparison of different values of k (rank of the feature matrices)

to achieve best performance for conventional matrix factorization [5, 16]. Figure 4(a) and 4(b) shows the performances of KMF using rbf kernel and MKMF on Jester dataset and Flixster dataset upon different values of k. The optimal rank for Jester dataset and Flixster dataset are both $k = 9$. It also shows that KMF and MKMF are overfitting while k increases. Other datasets show the similar pattern, denoting that both KMF and MKMF share the same principle with non-kernelized matrix factorization techniques on choosing parameter k. Note that in our experiments, we found the optimal k for ASSISTments dataset is about 120, which is not consist with the principle of $k \ll min(m, n)$. Since matrix factorization usually finds optimal low-rank feature matrices by choosing a relatively small value of k, it also explains the fact that all compared matrix factorization methods are outperformed by AVG. Because the ratings in ASSISTments dataset are all binary, it may make the data matrix contains some unique properties that difficult for low-rank matrix factorization methods to capture.

5 Related Works

Generally, collaborative filtering (CF) approaches can be classified into two categories: memory-based approach and model-based approach. In this section, we briefly discuss both of them.

5.1 Memory-based Approach The memory-based CF approach is also called neighbor-based CF, in which user-based methods and item-based methods are included. Neighbor-based CF method estimates the unobserved ratings of a target user or item as follows, it first find the observed ratings assigned by a set of neighboring users or the observed ratings assigned to a set of neighboring items, then it aggregates the neighboring ratings to derive the predicted rating of the target user or item. In order to find the neighboring set of a target user or item, similarity measures such as correlation-based similarity [19], vector cosine-based similarity [3] and conditional probability-based similarity [9] are usually used. Memory-based CF methods are usually easy to implement, and new data can be added into the model easily without re-training. But they are known to suffer from the sparsity problem which makes the algorithm hard to find highly similar neighboring sets. Several relaxation approaches were proposed to address the sparsity problem to fill in some of the unknown ratings using different techniques [15, 25]. Neighbor-based CF methods also have limited scalability for large datasets since each prediction is made by searching the similar users or items in the entire data space. When the number of users or items is large, the prediction is very expensive to obtain, which makes it difficult to scale in the online recommender system.

5.2 Model-based Approach Model-based approach can better address the sparsity problem than memory-based approach does, since it allows the system to learn a compact model that recognizes complex patterns based on the observed training data, instead of directly searching in rating database. Generally, classification algorithms can be employed as CF models for the dataset with categorical ratings, and regression models can be used if the ratings are numerical. Popular models used in this category include matrix factorization [16], probabilistic latent semantic analysis [7], Bayesian networks [17], clustering [4] and Markov decision process [23].

As to the matrix factorization approaches relevant to our work, [22] generalizes probabilistic matrix factorization to a parametric framework and requires the aid of topic models. [24] introduces a nonparametric factorization method under trace norm regularization. The optimization is cast into a semidefinite programming (SDP) problem, whose scalability is limited. A faster approximation of [24] is proposed by [18], which, in fact is very similar to the formulation of SVD [5]. [26] proposes a fast nonparametric matrix factorization framework using an EM-like algorithm that reduce the time cost on the large-scale dataset.

In this paper, we also take the model based approach considering its effectiveness. And we require no

aid from side information to make our proposed methods more generalized. To the best of our knowledge, this paper is the first work leverages both kernel methods and multiple kernel learning for matrix factorization to address the CF problem.

6 Conclusion

We have presented two novel matrix completion methods named KMF and MKMF, which both can exploit the underlying nonlinear correlations among rows (users) and columns (items) of the rating matrix simultaneously. KMF incorporates kernel methods for matrix factorization, which embeds the low-rank feature matrices into a much higher dimensional space, enabling the ability to learn nonlinear correlations upon the rating data in original space. MKMF further extends KMF to combine multiple kernels by learning the a set of weights for each kernel functions based on the observed data in rating matrix. As demonstrated in the experiments, our proposed methods improve the overall performance of predicting the unobserved data in the matrix compared to state-of-art baselines.

7 Acknowledgements

This work is supported in part by NSFC through grant NSFC(61403186).

References

[1] C. C. Aggarwal. *Recommender Systems: The Textbook*. Springer, 2016.

[2] F.R. Bach, G. Lanckriet, and M.I. Jordan. Multiple kernel learning, conic duality, and the smo algorithm. In *ICML*, page 6, 2004.

[3] J.S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *UAI*, pages 43–52, 1998.

[4] S. Chee, J. Han, and K. Wang. Rectree: An efficient collaborative filtering method. In *DaWaK*, pages 141–151. 2001.

[5] S. Funk. Netflix update: Try this at home, 2006.

[6] M. Gönen and E. Alpaydın. Multiple kernel learning algorithms. *The Journal of Machine Learning Research*, 12:2211–2268, 2011.

[7] T. Hofmann. Latent semantic models for collaborative filtering. *ACM Transactions on Information Systems*, 22(1):89–115, 2004.

[8] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM*, pages 263–272, 2008.

[9] G. Karypis. Evaluation of item-based top-n recommendation algorithms. In *CIKM*, pages 247–254, 2001.

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

[11] G. Lanckriet, T. De Bie, N. Cristianini, M.I. Jordan, and W.S. Noble. A statistical framework for genomic data fusion. *Bioinformatics*, 20(16):2626–2635, 2004.

[12] G. Lanckriet, N. Cristianini, P. Bartlett, L.E. Ghaoui, and M.I. Jordan. Learning the kernel matrix with semidefinite programming. *The Journal of Machine Learning Research*, 5:27–72, 2004.

[13] N.D. Lawrence and R. Urtasun. Non-linear matrix factorization with gaussian processes. In *ICML*, pages 601–608, 2009.

[14] C. Ma. A guide to singular value decomposition for collaborative filtering, 2008.

[15] H. Ma, I. King, and M.R. Lyu. Effective missing data prediction for collaborative filtering. In *SIGIR*, pages 39–46, 2007.

[16] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *KDD Cup*, pages 5–8, 2007.

[17] D.M. Pennock, E. Horvitz, S. Lawrence, and C.L. Giles. Collaborative filtering by personality diagnosis: A hybrid memory-and model-based approach. In *UAI*, pages 473–480, 2000.

[18] J. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *ICML*, pages 713–719, 2005.

[19] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *CSCW*, pages 175–186, 1994.

[20] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, pages 285–295, 2001.

[21] B. Schölkopf and A.J. Smola. *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

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

[23] G. Shani, R.I. Brafman, and D. Heckerman. An mdp-based recommender system. In *UAI*, pages 453–460, 2002.

[24] N. Srebro, J. Rennie, and T.S. Jaakkola. Maximum-margin matrix factorization. In *NIPS*, pages 1329–1336, 2004.

[25] G. Xue, C. Lin, Q. Yang, W. Xi, H. Zeng, Y. Yu, and Z. Chen. Scalable collaborative filtering using cluster-based smoothing. In *SIGIR*, pages 114–121, 2005.

[26] K. Yu, S. Zhu, J. Lafferty, and Y. Gong. Fast nonparametric matrix factorization for large-scale collaborative filtering. In *SIGIR*, pages 211–218, 2009.

[27] T. Zhou, H. Shan, A. Banerjee, and G. Sapiro. Kernelized probabilistic matrix factorization: Exploiting graphs and side information. In *SDM*, pages 403–414, 2012.