Worcester Polytechnic Institute

Kernel Methods for Collaborative Filtering

by

Xinyuan Sun

A thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Master of Science

in

Data Science

January 2016

APPROVED:

Professor Xiangnan Kong, Advisor:

_____

Professor Randy C. Paffenroth, Reader:

_____

Abstract

The goal of the thesis is to extend the kernel methods to matrix factorization(MF) for collaborative filtering(CF). In current literature, MF methods usually assume that the correlated data is distributed on a linear hyperplane, which is not always the case. The best known member of kernel methods is support vector machine (SVM) on linearly non-separable data. In this thesis, we apply kernel methods on MF, embedding the data into a possibly higher dimensional space and conduct factorization in that space. To improve kernelized matrix factorization, we apply multi-kernel learning methods to select optimal kernel functions from the candidates and introduce ℓ_2-norm regularization on the weight learning process. In our empirical study, we conduct experiments on three real-world datasets. The results suggest that the proposed method can improve the accuracy of the prediction surpassing state-of-art CF methods.

Contents

Chapter 1

Introduction

In this project, we applied multiple kernel matrix factorization(MKMF) for *collaborative filtering*. Conventionally, the typical application of multiple kernel learning(MKL) is to improve the performance of classification and regression tasks. The basic idea of MKL is combining multiple kernels instead of using a single one. The thesis incorporates the method to perform low-rank matrix factorization in a possibly much higher dimension feature space, learning nonlinear correlations toward the data in original space. Our goal is to improve the accuracy of prediction for collaborative filtering that serving the recommender systems. This chapter will give an overview about motivation, challenges as well as contributions of this work.

1.1 Motivation

The increasing demand on e-commerce brings out online data growing exponentially. A critical challenge for modern electronic retailers is to provide customers with personalized recommendations among large variety of offered products. *Collaborative filtering*(CF), which mainly serves for recommender systems making automatic predictions, is one of the technologies that address this challenge. Recommender systems are particularly useful for en-

tertainment products such as movies, music and TV shows. Some famous examples of recommender systems includes Amazon.com and Netflix.

Differed from other approaches, such as *content based models*, CF relies only on past user behavior, for example, previous transactions or browsing history, without requiring the creation of explicit profiles. One major appeal of CF is that it is domain free, yet it can address aspects of the data that are often elusive and very difficult to profile. Given how applicable it is, in recent years, CF has been widely researched and has achieved further progress with the extensive study on matrix factorization technologies.

In the basic form of matrix factorization, suppose we have a partially filled rating matrix, where each row stands for a user and each column denotes an item in the system, matrix factorization models map both users and items to a joint latent factor space of dimensionality f, such that user-item interactions - the user's overall interest in the item's charactaristics, are modeled as inner products in that space. As a result, the problem becomes predicting the missing entries with the inner product of latent feature vectors for user i and item j.

The existing works and models [4, 6, 16] for matrix factorization are mainly induced by Singular Value Decomposition(SVD) of the user-item observation matrix. SVD assumes that the ratings are distributed on a linear hyperplane which can be represented by the inner products of two low-rank structures. However, in many practical situations, the data of matrix $\mathbf{R}$ is nonlinearly distributed, which makes it hard or impossible to recover the full matrix by solving low-rank factorizations in linear way. 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. The inner products of two embedded items in the Hilbert feature space can be inferred by the kernel function, which is

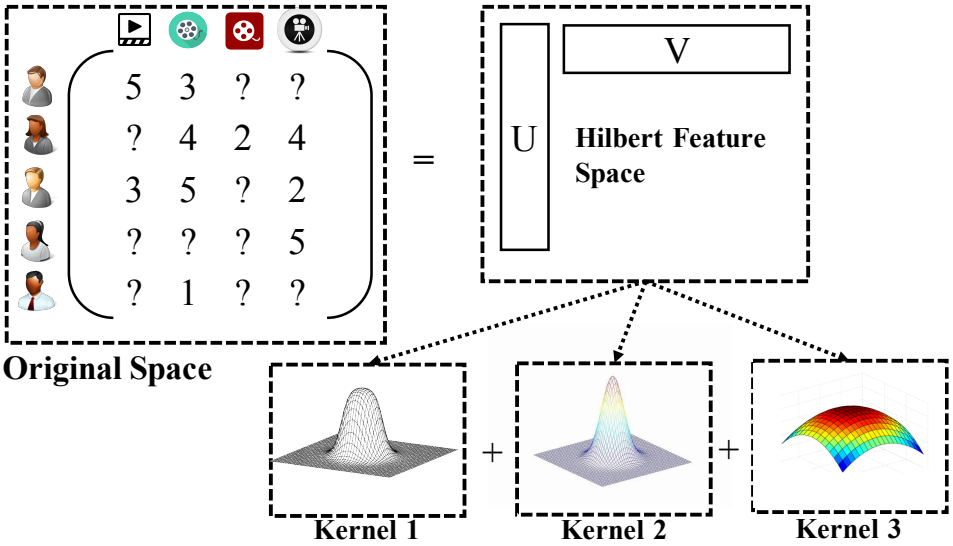


Figure 1.1: An illustration of kernelized low-rank matrix factorization for predicting the unobserved ratings. The two latent factor matrices $\mathbf{U}$ and $\mathbf{V}$ are embedded into a high-dimensional Hilbert feature space by a linear combination of kernel functions. The products of the two latent factor matrices reconstruct the rating matrix in original space nonlinearly.

also called the kernel trick.

Incorporating kernel methods into matrix factorization for collaborative filtering, however, is very challenging for the following three reasons: *First,* while the inner products of the embedding can be inferred by the kernel function, *e.g.,* for polynomial kernel, $\kappa(\mathbf{x_1}, \mathbf{x_2}) =< \phi(\mathbf{x_1}), \phi(\mathbf{x_2}) >= (\mathbf{x_1}^T \mathbf{x_2} + c)^d$, the mapping ϕ is usually implicitly defined. As a result, as to the kernerlized matrix factorization problems, the matrices of latent factors of users and items are implicitly defined. In that case, minimizing the objective function becomes more challenging since applying alternating least square(ALS) or gradient descent approaches is hampered by the implication of the latent factors. Thus, it is critical to devise effective kernel methods for the purpose of matrix completion.

Second, lacking side information. Existing kernelized matrix factoriza-

tion approaches [13, 28] 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.

Third, combining multiple kernels in terms of large scale collaborative filtering tasks. A kernel function can capture a certain notion of similarity and efficiently embed data into a much higher dimensional space, but using one single kernel may still lead to suboptimal performance. 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. Despite its value and significance, few effort has been made on combining multiple kernel learning and matrix factorization. Existing multiple kernel learning strategies are mostly designed for classification and regression models, such as SVM, ridge regression, *etc.*. Moreover, it is even more challenging when considering the kernel weight learning process with ℓ_2-norm considering large-scale problems.

To solve the above issues, we propose a solution, ℓ_2-norm MKMF, that combining matrix factorization with kernel methods and multiple kernel learning. Applying kernel methods on pure matrix factorization without side information, ℓ_2-norm MKMF can effectively capture the non-linear relationships in the rating data. Empirical studies on real-world datasets show that the proposed MKMF can improve the accuracy of prediction for collaborative filtering.

1.2 Contribution

In this thesis, we introduce a new matrix-factorization algorithm for collaborative filtering MKMF with ℓ_2-norm constraint on the kernel weight learning

process, which is based on multiple kernel learning and matrix factorization for collaborative filtering. We have implemented this algorithm on high dimensional datasets in real-world domain. A special optimization is known as a *quadratic constrained quadratic programming* is introduced regarding the kernel weight learning process to achieve better prediction accuracy on the rating data.

1.3 Structure of Thesis

The rest of the thesis is organized as follows. Chapter 2 describes the preliminary concepts and an overview of the problem. Chapter 3 introduces the proposed MKMF approach. Chapter 4 reports the experiments and results. Chapter 5 gives a brief introduction of the related works of collaborative filtering. Chapter 6 concludes this work with our advancements.

Chapter 2

Problem Statement

2.1 Notaions

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

We reserve special indexing letters for distinguishing users from items: for users u, and for items i. Each observed rating is a tuple (u, i, r_{ui}) denoting the preference by user u of item i, where the rating values $r_{ui} \in \mathbf{R}$. Notice that the matrix $\mathbf{R}$ contains $m \times n$ entries, where m denotes the number of users and n is the number of items. Each user is assumed to rate an item only once, so all such ratings can be arranged into an $m \times n$ matrix $\mathbf{R}$ whose ui-th entry equals r_{ui}. The missing entires in $\mathbf{R}$ corresponding to the unobserved rating, which is what we try to make prediction. The observed entries indexed by the set $\Omega = \{(u, i) : r_{ui} \}$ is observed, and the number of the observed ratings is $|\Omega|$. Typically $|\Omega|$ is much smaller than $m \times n$ because of the sparsity problem.

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;

- $\Omega_u = (1, 3)^T$ and $\mathbf{r}_{\Omega_u} = (r_{u1}, r_{u3})^T$.

- 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 $\mathbf{r}_{:,i}$'s ratings are all unobserved except the 2nd and the 4th values, then;

- $\Omega^i = (2, 4)^T$ and $\mathbf{r}_{:,\Omega^i} = (r_{2i}, r_{4i})^T$.
- Given a matrix $\mathbf{A}$, $\mathbf{A}_{:,\Omega_u}$ 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 $\{r_{ui}|(u, i) \notin \Omega\}$ based on the observed ratings.

Table 2.1: 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 i-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
$\{\mathbf{d}_1, ..., \mathbf{d}_k\}$	The set of dictionary vectors
$\phi(.)$	The implicit mapping function of some kernel
$\Phi = (\phi(\mathbf{d}_1), ..., \phi(\mathbf{d}_k))$	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 for items
$\{\kappa_1, ..., \kappa_p\}$	The set of base kernel functions
$\{\mathbf{K}_1, ..., \mathbf{K}_p\}$	The set of base kernel matrices

2.2 Matrix Factorization

Matrix factorization is one of the most successful realizations of collaborative filtering. The idea of matrix factorization is to approximate the observed matrix R as the product of two low-rank matrices:

$$\mathbf{R} \approx \mathbf{U}^T\mathbf{V}$$

where $\mathbf{U}$ is a $k \times m$ matrix and $\mathbf{V}$ is a $k \times n$ matrix. Here k stands for 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 $k \ll min(m, n)$.

The model map both users and items to a joint latent factor space of dimensionality f. Accordingly, each user u is associated with a vector $\mathbf{u}_u \in \mathbb{R}^f$, and each item i is associated with a vector $\mathbf{v}_i \in \mathbb{R}^f$. The prediction for rating assigned to item i by user u, as mentioned before, is done by taking an inner product:

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

where u_{ju} denotes the 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 v_i denotes the i-th column of $\mathbf{V}$.

Modern systems modeling directly only the observed ratings, while avoiding overfitting through a regularization term. 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 [7,10] 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 since it is a local convex problem eventually leads to the solution of $\hat{\mathbf{U}}$ and $\hat{\mathbf{V}}$.

2.3 ℓ_2-norm multiple kernel learning

Since the right choice of kernel is usually unknown, applying single kernel function may still lead to suboptimal results. As a result, multiple kernel learning(MKL) works to learn an optimal linear combination of kernels $\mathbf{K} = \sum_{j=1}^m \mu_j \mathbf{K_j}$ with non-negative coefficients $\boldsymbol{\mu}$ together with the model parameters. One conventional approach to multiple kernel learning employ ℓ_1-norm constraints on the mixing coefficients $\boldsymbol{\mu}$ to promote sparse kernel combinations. However, when features encode orthogonal characterizations of a problem, sparseness may lead to discarding useful information and may thus result in poor generalization performance [9]. Thus, we impose ℓ_2-norm constraint on the mixing coefficients $\boldsymbol{\mu}$ that leads to a *quadratic constrained*

quadratic programming(QCQP). QCQP has the basic form as

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\boldsymbol{\mu}^\top \mathbf{P_0}\boldsymbol{\mu} + \mathbf{q_0}^T\boldsymbol{\mu} \\ \text{subject to} \quad & \frac{1}{2}\boldsymbol{\mu}^\top \mathbf{P_i}\boldsymbol{\mu} + \mathbf{q_i}^T\boldsymbol{\mu} + r_i \leq 0 \quad \text{for } i = 1, ..., m, \\ & \mathbf{A}\boldsymbol{\mu} = b \end{aligned} \tag{2.5}$$

where $P_0, ..., P_m$ are n-by-n matrices and $\boldsymbol{\mu} \in \boldsymbol{R}^n$ is the optimization variable. The optimization problem is considered convex when $P_0, ..., P_m$ are all positive semi-definite. In this thesis, the kernel weight learning process using ℓ_2-norm regularization also solved as the QCQP convex problem.

Chapter 3

Multiple Kernel Collaborative Filtering

Traditional matrix factorization recover the full matrix by solving low-rank factorization linearly. In this chapter, we introduce the kernel methods that aim to add non-linearity to MK problem assuming the collaborative filtering data is distributed on a nonlinear hyperplane.

3.1 Kernels

Kernel methods work for turned linear model into a non-linear model by embedding data into much higher dimensional (possibly infinite), implicit feature space. It follows the Mercer's theorem that an implicitly defined function ϕ exists whenever the space χ can be equipped with a suitable measure ensuring the kernel function K satisfies mercer's condition. While the embedding is implicit here, the inner product of data point in the feature space can be inferred, such that $\phi(\mathbf{x}_1)^T\phi(\mathbf{x}_2) = \kappa(\mathbf{x}_1, \mathbf{x}_2) \in \mathbf{R}$, where κ denotes the kernel function of the corresponding kernel. For example, for polynomial kernel with degree-d, $\kappa(\mathbf{x_1}, \mathbf{x_2}) =< \phi(\mathbf{x_1}), \phi(\mathbf{x_2}) >= (\mathbf{x_1}^T\mathbf{x_2} + c)^d$. 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

One challenge for incorporating kernel methods into matrix factorization, as mentioned in Chapter 1, is the implicity of the latent factors of users and items. To address the challenge, we use the dictionary-vectors for the computing of Gram matrix.

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:

$$\phi(\mathbf{u}_u) = \sum_{j=1}^{k} a_{uj}\phi(\mathbf{d}_j) = \mathbf{\Phi}\mathbf{a}_u, \tag{3.1}$$

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:

$$\phi(\mathbf{v}_i) = \sum_{j=1}^{k} b_{ij}\phi(\mathbf{d}_j) = \mathbf{\Phi}\mathbf{b}_i, \tag{3.2}$$

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 $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$

$$\underset{\phi(\mathbf{v}_i)\in\mathcal{H}}{\text{minimize}} \sum_{u} (r_{ui} - \phi(\mathbf{u}_u)^\top \phi(\mathbf{v}_i))^2 + \lambda\phi(\mathbf{v}_i)^\top\phi(\mathbf{v}_i) \tag{3.3}$$

It is easy to see

$$\phi(\mathbf{u}_u)^\top \phi(\mathbf{v}_i) = \mathbf{a}_u^\top \mathbf{\Phi}^\top \mathbf{\Phi} \mathbf{b}_i = \mathbf{a}_u^\top \mathbf{K} \mathbf{b}_i, \tag{3.4}$$

$$\phi(\mathbf{v}_i)^\top \phi(\mathbf{v}_i) = \mathbf{b}_i^\top \mathbf{\Phi}^\top \mathbf{\Phi} \mathbf{b}_i = \mathbf{b}_i^\top \mathbf{K} \mathbf{b}_i, \tag{3.5}$$

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 (3.3) as

$$\underset{\mathbf{b}_i \in \mathbb{R}^k}{\text{minimize}} \sum_u (r_{ui} - \mathbf{a}_u^\top \mathbf{K} \mathbf{b}_i)^2 + \lambda \mathbf{b}_i^\top \mathbf{K} \mathbf{b}_i \tag{3.6}$$

which is equivalent to

$$\underset{\mathbf{b}_i \in \mathbb{R}^k}{\text{minimize}} \; ||\mathbf{r}_{:,\Omega^i} - \mathbf{A}_{:,\Omega^i}^\top \mathbf{K} \mathbf{b}_i||_F^2 + \lambda \mathbf{b}_i^\top \mathbf{K} \mathbf{b}_i, \tag{3.7}$$

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

$$\hat{\mathbf{b}}_i = (\mathbf{K}^\top \mathbf{A}_{:,\Omega^i} \mathbf{A}_{:,\Omega^i}^\top \mathbf{K} + \lambda \mathbf{K})^\dagger \mathbf{K}^\top \mathbf{A}_{:,\Omega^i} \mathbf{r}_{:,\Omega^i} \tag{3.8}$$

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 (3.7). In this case, the closed form solution for each $\hat{\mathbf{a}}_u$ is given by:

$$\hat{\mathbf{a}}_u = (\mathbf{K}^\top \mathbf{B}_{:,\Omega_u} \mathbf{B}_{:,\Omega_u}^\top \mathbf{K} + \lambda \mathbf{K})^\dagger \mathbf{K}^\top \mathbf{B}_{:,\Omega_u} \mathbf{r}_{\Omega_u} \tag{3.9}$$

3.3 Multiple Kernel Matrix Factorization

Multiple kernel learning(MKL) [1, 5, 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 definte 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

Algorithm 1 Multiple Kernel Matrix Factorization

Require: $k, d, \{\kappa_1, \ldots, \kappa_p\}, \mathbf{R}, \Omega, \lambda, iter_{max}$

allocate $\mathbf{A} \in \mathbb{R}^{k \times m}, \mathbf{B} \in \mathbb{R}^{k \times n}, \mu \in \mathbb{R}^p, \mathcal{D} = \{\mathbf{d}_i \in \mathbb{R}^d : 1 \leq i \leq k\}, \{\mathbf{K}_i \in \mathbb{R}^{k \times k} : 1 \leq i \leq p\}, \mathbf{K} \in \mathbb{R}^{k \times k}$

initialize $\mu = (\frac{1}{p}, \ldots, \frac{1}{p})^\top$, $\mathbf{A}$, $\mathbf{B}$, $\mathcal{D}$

for $i \leftarrow 1, p$ **do**

    $\mathbf{K}_i \leftarrow (\kappa_i(\mathbf{d}_h, \mathbf{d}_j))_{1 \leq h,j \leq k}$

end for

$iter \leftarrow 0$

repeat

    $\mathbf{K} \leftarrow \sum_{i=1}^{p} \mu_i \mathbf{K}_i$

    Update $\mathbf{B}$ as shown in (3.8)

    Update $\mathbf{A}$ as shown in (3.9)

    Update μ by solving (3.11)

until $iter = iter_{max}$ or convergence

Return $\mathbf{A}, \mathbf{B}, \mu, \mathbf{K}$

$\mu = (\mu_1, \ldots, \mu_p)^\top$. The learning task can be cast into following optimization:

$$\begin{aligned} \underset{\mathbf{a}_u, \mathbf{b}_i \in \mathbb{R}^k}{\text{minimize}} \sum_{(u,i)\in\Omega} (r_{ui} - \underbrace{\sum_{j=1}^{p} \mu_j \mathbf{a}_u^\top \mathbf{K}_j \mathbf{b}_i}_{\boldsymbol{\upsilon}_{ui}^\top \boldsymbol{\mu}})^2 + \\ \underbrace{\lambda(\mathbf{b}_i^\top \sum_{j=1}^{p} \mu_j \mathbf{K}_j \mathbf{b}_i + \mathbf{a}_u^\top \sum_{j=1}^{p} \mu_j \mathbf{K}_j \mathbf{a}_u)}_{\boldsymbol{\gamma}_{ui}^\top \boldsymbol{\mu}} \end{aligned} \tag{3.10}$$

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 (3.10),

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{\mu}^\top \mathbf{Y} \boldsymbol{\mu} + \mathbf{Z} \boldsymbol{\mu} \\ \text{subject to} \quad & \boldsymbol{\mu} \succeq 0 \\ & \|\boldsymbol{\mu}\|_2 \leq 1 \end{aligned} \tag{3.11}$$

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, we impose ℓ_2-norm constraint on $\boldsymbol{\mu}$, make the optimization problem (3.11) is known as a *quadratic constrained quadratic programming*. Unlike ℓ_1-norm regularization,

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{\mu}^\top \mathbf{Y} \boldsymbol{\mu} + \mathbf{Z} \boldsymbol{\mu} \\ \text{subject to} \quad & \boldsymbol{\mu} \succeq 0 \\ & \mathbf{1}_p^\top \boldsymbol{\mu} = 1 \end{aligned} \tag{3.12}$$

which is a *quadratic programming* optimization problem, the ℓ_2-norm regularization is more challenging, but it can also be solved in principle by general purpose optimization toolboxes such as MOSEK[1]. Notice that, we constraint $\boldsymbol{\mu} \succeq 0$ to make sure the kernels are positive definite.

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

[1] https://www.mosek.com/

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 $\mathbf{K}_i = (\kappa_i(\mathbf{d}_h, \mathbf{d}_j))_{1 \leq h,j \leq k}$. 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 $\mathbf{K}^0 = \sum_{1 \leq i \leq p} \mu_i^0 \mathbf{K}_i$. 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 (3.7), each solution can be obtained directly by computing the closed form expression shown in 3.8. 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 3.12 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: $\quad \hat{r}_{ui} \leftarrow \overset{*}{\mathbf{a}}_u^\top \overset{*}{\mathbf{K}} \overset{*}{\mathbf{b}}_i$
4: $\quad$ add $\hat{r}_{ui}$ to $\mathcal{P}$
5: **end for**
6: **Return** $\mathcal{P}$.

Chapter 4

Experiments

4.1 Datasets

The proposed method is experimented and evaluated on 3 real-world collaborative filtering datasets: MovieLens, Jester and Flixster. The details are summarized in Table 4.1. 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 Flixster dataset contains only 5 rating classes from 1 to 5. 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 (Jester dataset only contains 100 items).

Table 4.1: Summary of compared methods.

Dataset	# of users	# of items	Density (%)	Rating range
MovieLens	6,040	3,900	6.3	[1,5]
Jester	73,412	100	55.8	[-10.0,10.0]
Flixster	147,612	48,784	0.11	[1,5]

Table 4.2: 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	[4]
MF	Model-Based	No	[10]
KMF	Model-Based	Single Kernel	This thesis
MKMF	Model-Based	Multiple Kernels	This thesis

4.2 Baselines

We compare our proposed method, ℓ_2-norm MKMF, with multiple sets of baselines, among which AVG, IVS-COS and IVC-PERSON are memory-based methods while SVD, MF, KMF and MKMF are model-based methods.

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

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

2) IVC-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].

3) IVC-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.

4) SVD: A model based method called Funk-SVD [4]. 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.

5) MF: textscMf (Matrix Factorization) is a model-based CF method that discussed in Section 2.2.

6) 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.

7) ℓ_1-norm MKMF: MKMF that employ ℓ_1-norm constraints on the weight coefficients. The 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 3.3.

8) ℓ_2-norm MKMF: The proposed MKMF that imposing an ℓ_2-norm constraint on the weight coefficients.

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

The performance metric for rating accuracy used here is the root mean square error (RMSE):

$$RMSE = \sqrt{\frac{\sum_{(u,i)\in\Omega}(r_{ui} - \hat{r}_{ui})^2}{|\Omega|}} \tag{4.1}$$

4.4 Settings

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 (3.10). The proposed KMF method is also tested using the three base kernels respectively.

Empirical results on the three real-word datasets are summarized in Table 4.3 and 4.4.

Table 4.3: Results (RMSE(rank)) of `Leave 1` on the real-world datasets.

Methods	Flixster		MovieLens		Jester		Ave. Rank
ℓ_2-norm MKMF	0.8274	(2)	0.8233	(2)	4.0515	(5)	3
ℓ_1-norm MKMF	0.8270	(1)	0.8223	(1)	4.0398	(1)	1
KMF(LINEAR)	0.8290	(5)	0.8241	(4.5)	4.0471	(2.5)	4
KMF(POLY)	0.8289	(4)	0.8241	(4.5)	4.0472	(4)	4.1667
KMF(RBF)	0.8292	(6)	0.8246	(6)	4.0471	(2.5)	4.8333
MF	0.8286	(3)	0.8235	(3)	4.0549	(6)	4
SVD	0.9441	(10)	0.9406	(8)	4.2794	(7)	8.3333
IVC-COS	0.9223	(8)	1.0016	(9)	4.6137	(9)	9
IVC-PEARSON	0.9226	(9)	1.0020	(10)	4.6142	(10)	9.6667
AVG	0.9006	(7)	0.8887	(7)	4.3867	(8)	7.3333

Table 4.4: Results (RMSE(rank)) of `Leave 3` on the real-world datasets.

Methods	Flixster		MovieLens		Jester		Ave. Rank
ℓ_2-norm MKMF	0.8213	(6)	0.8201	(6)	4.0883	(6)	6
ℓ_1-norm MKMF	0.8153	(1)	0.8168	(1)	4.0809	(1)	1
KMF(LINEAR)	0.8163	(3.5)	0.8178	(2.5)	4.0826	(3)	3
KMF(POLY)	0.8163	(3.5)	0.8179	(4)	4.0824	(2)	3.1667
KMF(RBF)	0.8157	(5)	0.8186	(5)	4.0829	(4)	4.6667
MF	0.8155	(2)	0.8178	(2.5)	4.0865	(5)	3.1667
SVD	0.9335	(10)	0.9346	(8)	4.2601	(7)	8.3333
IVC-COS	0.9050	(8)	1.0034	(9)	4.6282	(9)	8.6667
IVC-PEARSON	0.9051	(9)	1.0047	(10)	4.6284	(10)	9.6667
AVG	0.8868	(7)	0.8865	(7)	4.3656	(8)	7.3333

Based on the results, we have made following interesting observations:

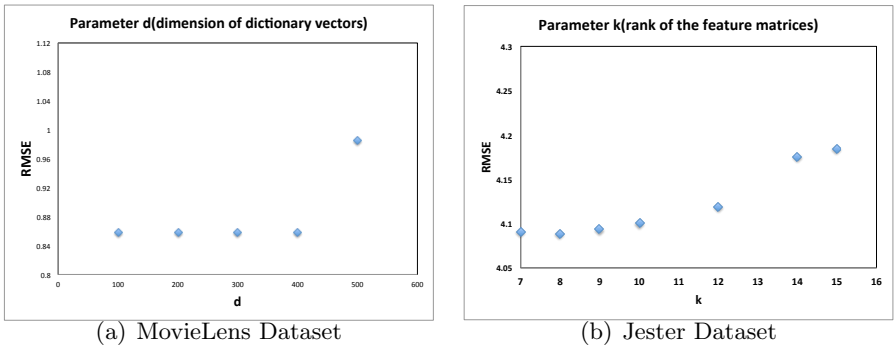
- The proposed ℓ_2-norm MKMF generally outperforms the other baselines, except ℓ_1-norm MKMF, on two of the three datasets in `Leave 1`; The result of ℓ_1-norm MKMF, on the other hand, surpassing all other methods in both `Leave 1` and `Leave 3`. Such experimental result shows that incorporating kernels into matrix factorization helps the model capturing the non-linear correlation among the data.

- In regard to self-comparison, the performance of both ℓ_1-norm MKMF and ℓ_2-norm MKMF is 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.

- Notice that ℓ_2-norm MKMF outperformed by ℓ_1-norm MKMF in all cases. Though generally ℓ_2-norm regularization provides more precise solution, in matrix factorization problem, the sparse kernel combination using ℓ_1-norm regularization proves to be more robust. While the test performance is less ideal than what we expected, the training RMSE of ℓ_2-norm MKMF remains to be low in both two settings.

4.6 Parameter Studies

Figure 4.1(a) shows the kernelized methods' sensitivity of parameter d, which stands for the dimensions of the dictionary vectors. Generally, according to the figures, MKMF are not very sensitive to the value of d once we choose relatively large d. For MovieLens dataset, the optimal choice of d is around 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



(a) MovieLens Dataset (b) Jester Dataset

Figure 4.1: Parameter studies over d and k

the insensitive pattern is shown in Figure 4.1(a).

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 to achieve best performance for conventional matrix factorization [4, 16]. Figure 4.1(b) shows the performances of MKMF on Jester dataset upon different values of k. The optimal rank for Jester dataset is $k = 8$. It also shows that MKMF are overfitting while k increases.

Chapter 5

Related Works

In current research, collaborative filtering(CF) works in one of two areas: memory-based approach and model-based approach. In this section, we introduces both two approaches briefly.

5.1 Memory-based Approaches

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 [2] and conditional probability-based similarity [8] 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, 26]. 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 Approaches

Model-based approach is an alternative approach that 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 [6], Bayesian networks [17], clustering [3] 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 [4]. [27] proposes a fast nonparametric matrix factorization framework using an EM-like algorithm that reduce the time cost on the large-scale dataset.

Chapter 6

Conclusion

In this thesis, we have introduced a matrix factorization model, ℓ_2-norm MKMF, for collaborative filtering. The proposed model can capture the non-linear distribution of latent factor effectively. The ℓ_2-norm MKMF extends ℓ_1-norm MKMF to impose ℓ_2-norm constraint on the weight coefficients regarding kernel weight learning. The ℓ_2-norm constraint on the constraint result in a QCQP optimization. As demonstrated in the experiments, our proposed method improve the accuracy of prediction and surpassing the results of multiple state-of-art collaborative filtering methods.

Bibliography

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

[2] J.S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 43–52, 1998.

[3] S. Chee, J. Han, and K. Wang. Rectree: An efficient collaborative filtering method. In *Data Warehousing and Knowledge Discovery*, pages 141–151. Springer, 2001.

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

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

[6] T. Hofmann. Latent semantic models for collaborative filtering. *TOIS*, 22(1):89–115, 2004.

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

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

[9] Marius Kloft, Ulf Brefeld, Pavel Laskov, and Sören Sonnenburg. Non-sparse multiple kernel learning. In *NIPS Workshop on Kernel Learning: Automatic Selection of Optimal Kernels*, volume 4, 2008.

[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. ACM, 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. ACM, 2007.

[16] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD cup and workshop*, 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 *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 473–480, 2000.

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

[19] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186. ACM, 1994.

[20] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, pages 285–295. ACM, 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. IEEE, 2010.

[23] G. Shani, R.I. Brafman, and D. Heckerman. An mdp-based recommender system. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 453–460. Morgan Kaufmann Publishers Inc., 2002.

[24] N. Srebro, J. Rennie, and T.S. Jaakkola. Maximum-margin matrix factorization. In *Advances in neural information processing systems*, pages 1329–1336, 2004.

[25] X.Liu, C.Aggarwal, Y.Li, X.Kong, X.Sun, and S.Sathe. Kernelized matrix factorization for collaborative filtering. SDM, 2016.

[26] 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. ACM, 2005.

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

[28] T. Zhou, H. Shan, A. Banerjee, and G. Sapiro. Kernelized probabilistic matrix factorization: Exploiting graphs and side information. In *SDM*, volume 12, pages 403–414. SIAM, 2012.