# Kernel-Based Feature Extraction For Collaborative Filtering

Saket Sathe*, Charu C. Aggarwal*, Xiangnan Kong†, Xinyue Liu†
*IBM T. J. Watson Research Center, Yorktown Heights, New York 10598, USA.
Email: {ssathe, charu}@us.ibm.com
†Computer Science Department, Worcester Polytechnic Institute, USA.
Email: {xkong, xliu4}@wpi.edu

***Abstract*—Singular value decomposition (SVD) has been used widely in the literature to recover the missing entries of a matrix. The basic principle in such methods is to assume that the correlated data is distributed with a low-rank structure. The knowledge of the low-rank structure is then used to predict the missing entries. SVD is based on the assumption that the data (user ratings) are distributed on a linear hyperplane. This is not always the case, and the data could often be distributed on a nonlinear hyperplane. Therefore, in this paper, we explore the methodology of *kernel feature extraction* to complement off-the-shelf methods for improving their accuracy. The extracted features can be used to enhance a variety of existing methods such as biased matrix factorization and SVD++. We present experimental results illustrating the effectiveness of using this approach.**

## I. Introduction

A well-known method for making recommendations using collaborative filtering is the latent factor approach [1]. The latent factor approach implicitly assumes that the ratings *can be projected on a low-dimensional linear hyperplane*. The given ratings are used for estimating this hyperplane and their low-dimensional projections on this hyperplane. This projected representation provides a robust estimation of the unobserved ratings. The key idea behind the success of the latent factor models is that the lower dimensional hyperplane, on which the data is projected, can be estimated using an incomplete ratings matrix. This approach will, however, not be effective, when the data does not naturally align along a low-dimensional *linear* hyperplane. In many cases, the matrix can be better projected on *nonlinear* manifolds. Therefore, one approach is to formulate the problem such that the latent factors are in a nonlinear kernel space. This approach often overfits the sparsely specified ratings matrix and further worsens the quality of the underlying solution.

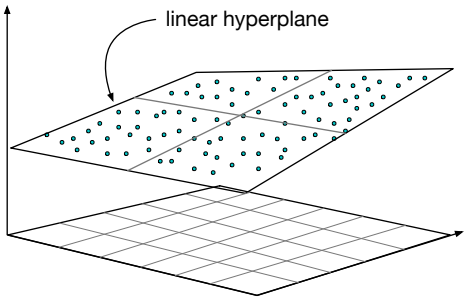
In many settings, it is more useful to leverage kernel methods for *feature extraction* rather than directly solving a particular problem with the kernel trick. Feature extraction provides many benefits in terms of combining the extracted features in an additive way with methods that are already known to be successful for particular data sets. In almost all cases that we tested, the feature extraction approach was able to improve the performance of a base method, such as biased matrix factorization and SVD++. Therefore, if one can determine the particular approach, which is suitable for a given data set (with cross-validation), the feature extraction methodology will still be able to enhance its accuracy. This type of approach provides the flavor of a feature extraction-based meta-algorithm. However, the process of combining the extracted features with a specific method requires some additional tricks. For example, in the case of matrix factorization, the extracted features are used to fix one of the factors and extract the other set of features. This avoids the natural tendency of kernel methods to overfit, and is therefore *complementary* to traditional matrix factorization models. Likewise, the approach can be used to enhance several off-the-shelf matrix factorization models with modest modifications. In the particular case of factorization methods and their variants, it is relatively easy to use any kernel of choice for deriving a new set of features. As specific examples, we will show the applicability of this approach to biased matrix factorization and SVD++, and an extended version of this paper discusses its application to item-based neighborhood models.

Our kernel-based methods are focused on extracting *item* features. As a complementary approach, one can also extract analogous *user* features. However, we did not pursue this approach because we found that (a) the approach was too computationally intensive because the number of users is often much greater than the number of items; (b) in a recommendation system, users have more churn than items, and, therefore, the resulting model becomes less relevant with time; (c) our preliminary tests showed that item factors always provided better accuracy. Therefore, we have omitted this complementary approach from this paper.

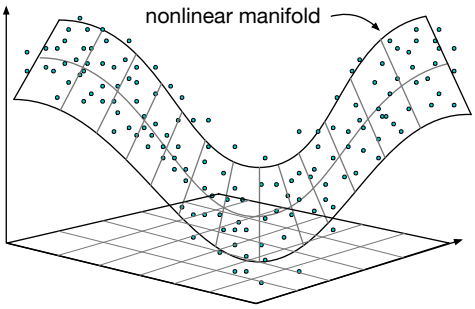
A related work in [3] proposes a *specific* kernel-based method, but it cannot be used to improve the effectiveness of a particular matrix factorization model. The approach proposed in [3] is sometimes outperformed by existing techniques because different methods perform more effectively on different data sets. Therefore, unlike the work in [3], our work has the flavor of a general meta-algorithm and has the ability to consistently improve collaborative filtering models.

## II. Background and Motivation

The basic idea of all latent factor models is to recover a low-rank structure. In this paper, we explore a popular low-rank latent factor method known as the SVD model. Imagine a data set where the ratings are distributed on a low-dimensional




(a) Fitting ratings with a linear hyperplane




(b) Fitting ratings with a nonlinear manifold

Figure 1. Linear and non-linear manifolds in 3-d space.

hyperplane as shown in Figure 1(a). It is possible to use SVD to discover this low dimensional plane, even when the data is incompletely specified. Assume that we have $m$ users and $n$ items. Let $R = [r_{ij}]$ be an $m \times n$ ratings matrix, which is incompletely specified. Let $E$ be the set of entries in the $m \times n$ ratings matrix $R$ that are specified.

$$E = \{(i, j) : r_{ij} \text{ is specified}\} \tag{1}$$

We want to factorize $R \approx UV^T$. Note that $U = [u_{ij}]$ is an $m \times k$ matrix, whereas $V = [v_{ij}]$ is an $n \times k$ matrix. Here, $k$ is the rank of the factorization. Such a factorization is analogous to the $k$-rank SVD factorization given as $R \approx Q_k \Sigma_k P_k^T$. Here, $U$ is analogous to the matrix $Q_k$, and $V$ is analogous to the matrix $P_k$ in *SVD*. The diagonal matrix $\Sigma_k$ can be absorbed in either $U$ or $V$, as long as the columns of $U$ and $V$ are orthogonal. Therefore, the optimum solution is not unique in terms of how $U$ and $V$ are scaled. One can use the factored matrices to predict the $(i, j)$th rating as follows:

$$\hat{r}_{ij} = \sum_{s=1}^{k} u_{is} v_{js}. \tag{2}$$

The relevant optimization problem is computed by minimizing the squared prediction error on the observed entries:

$$\text{Minimize } J = \frac{1}{2} \sum_{(i,j) \in E} \left( r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js} \right)^2 + \frac{\lambda_f}{2} \sum_{i,s} u_{is}^2 + \frac{\lambda_f}{2} \sum_{j,s} v_{js}^2.$$

Here $\lambda_f$ is a regularization parameter. The objective function is the same as the Frobenius norm of the error matrix, but computed only on the specified entries in $E$. The value of $UV^T$ provides the rank-$k$ projection of all user ratings on this hyperplane, which corresponds to the relevant estimate of the missing entries. This approach works well for the case of Figure 1(a). However, when the data is not linearly distributed on a low-dimensional hyperplane, as shown in Figure 1(b), using straightforward matrix factorization will not be robust. In such cases, a *nonlinear* manifold, as shown in Figure 1(b), is more suitable for accurately modeling the ratings.

### A. Broad Contours of Approach

Traditional matrix factorization derives $U$ and $V$ *simultaneously* with the use of gradient descent methods. A straightforward approach with kernels is not a simple matter in these settings. This is because if we transform the ratings matrix $R$ to the kernel feature space, there is usually no way of recovering both the user and item factors with the use of kernel methods. Kernel PCA allows us to only extract the embedding (user factors), but not the basis vectors (item factors). One can apply kernel PCA to the transpose of $R$, but then one can extract the embedding of the items (item factors), but not the basis vectors in the user space (user factors). Therefore, either the user factors or the item factors cannot be recovered if we assume that the ratings are transformed to a high dimensional space. Therefore, we decouple the process of determining the item and user factors. The first step is to use kernel PCA to discover a robust embedding of the items in a new feature space. This embedding is naturally defined with respect to a high-quality feature space because of the ability of kernel PCA to adapt to the nonlinear characteristics of the data.

Subsequently, the goal is to use matrix factorization to discover a similar embedding of the users in this feature space (user factors), so that the *original ratings matrix* can be expressed as a product of the user factor matrix and the item factor matrix. Therefore, kernel methods *are used to fix the feature space* in which the factorization is performed. As a result, one only needs to learn the user factors with gradient descent; the item factors are computed using kernel PCA and are already fixed. This type of approach has the added benefit of reducing the overfitting inherent to kernel methods because it does not attempt to learn the kernel features and user factors jointly. Furthermore, the approach is relatively easy to adapt to virtually any matrix factorization method. In the following description, however, we will focus on biased matrix factorization and SVD++.

## III. Kernel Latent Factor Models

The kernel latent factor models that we propose in this paper first use kernel *PCA* to extract a robust set of item features. Subsequently, the users are expressed with respect to the kernel item features in order to factorize the matrix. Therefore, the overall approach can be described as follows:

1) Use kernel PCA to extract the $k$-dimensional item features for each of the $n$ items. This results in an $n \times k$ item factor matrix $V_0$.
2) Factorize the original ratings matrix as $R \approx UV_0^T$. Note that $V_0$ is fixed in this factorization and only the ratings matrix $U$ needs to be learned. The gradient descent steps for the resulting factorization are much simpler and robust because one only needs to learn the user factors in gradient descent.

In the following subsections, we will describe each of these steps in detail.

### A. Learning the Item Features with Kernel Methods

The first step is that of extracting the kernel features of each item. This can be achieved by performing kernel PCA of the transpose of the ratings matrix. However, the ratings matrix is first adjusted for biases. Then, the kernel function is used to construct an $n \times n$ item-item kernel similarity matrix. This matrix is centered, and the $k$-dimensional embedding of each item is extracted. The approach works as follows:

1) **Bias Estimation and Removal:** In this step, the user and item biases are removed from each rating $r_{ij}$ in order to facilitate extraction of better features that are based only on personalized interactions. The biases are first estimated using a rudimentary model of the user bias ($b_i^{user}$) and item bias ($b_j^{item}$) that are associated with each user $i$, and item $j$. The prediction model used is as follows:

$$\hat{r}_{ij} = b_i^{user} + b_j^{item}. \tag{3}$$

The results in the following optimization formulation, which is designed to learn only the bias variables:

$$\text{Minimize } J = \frac{1}{2} \sum_{(i,j) \in E} (r_{ij} - \hat{r}_{ij})^2 + \frac{\lambda_b}{2} \left( \sum_i (b_i^{user})^2 + \sum_j (b_j^{item})^2 \right).$$

The stochastic gradient-descent updates to solve for the bias variables are as follows:

$$b_i^{user} \leftarrow b_i^{user} + \alpha_b (r_{ij} - \hat{r}_{ij} - \lambda_b b_i^{user})$$
$$b_j^{item} \leftarrow b_j^{item} + \alpha_b (r_{ij} - \hat{r}_{ij} - \lambda_b b_j^{item})$$

These iterations are executed to convergence. The estimated user and item biases are subtracted from their corresponding rating and an unbiased ratings matrix $R_u$ is computed.

2) **Kernel Matrix Construction:** Generate an $n \times n$ kernel matrix $S$ from $R_u$, which is obtained by computing kernel similarity between each pair of items (rows of $R_u^T$). Any off-the-shelf kernel function may be used such as a Gaussian kernel. The similarity between a pair of columns $\overline{rc_i}$ and $\overline{rc_j}$ of $R_u$ (i.e., rows of $R_u^T$) is computed as follows:

$$K(\overline{rc_i}, \overline{rc_j}) = \exp\left( -\frac{||\overline{rc_i} - \overline{rc_j}||^2}{2\sigma^2} \right). \tag{4}$$

For the purpose of kernel feature computation, unobserved entries are set to 0 after bias removal. The $(i, j)$th entry of $S$ is given by $K(\overline{rc_i}, \overline{rc_j})$. Note that the kernel matrix is of size $n \times n$. This kernel matrix is then mean centered as follows:

$$S \leftarrow (I - O/n)S(I - O/n). \tag{5}$$

Here $O$ is an $n \times n$ matrix containing only 1s, and $I$ is an $n \times n$ identity matrix. By the property of kernels, $S$ is a symmetric positive semi-definitive matrix with nonnegative eigenvalues.

3) **Item Embedding Extraction:** Extract a rank-$k$ embedding $V_0$ from $S$ by using the top-$k$ eigenvectors of $S$ as $V_0 = Q\Sigma$, where $S \approx Q\Sigma^2 Q^T = (Q\Sigma)(Q\Sigma)^T = V_0 V_0^T$. Therefore, $S$ is the dot-product matrix $V_0 V_0^T$ in terms of the transformed representation $V$ of the items. Note that the columns of $Q$ contain the dominant eigenvectors of $S$ and $\Sigma$ is a diagonal matrix containing the square root of the nonnegative eigenvalues. $V_0$ is an $n \times k$ matrix in which each row corresponds to the embedded representation of an item.

In typical settings, the number of items is smaller than the number of users, and therefore the kernel matrix is often of manageable size. The kernel bandwidth $\sigma$ is estimated by sampling a predefined number of $P$ pairs of items from the matrix $R_u$ and computing

$$\sqrt{\frac{\sum_{(i,j) \in P} \|\overline{rc_i} - \overline{rc_j}\|^2}{P}}. \tag{6}$$

As a rule-of-thumb this gives a good estimate of the kernel bandwidth $\sigma$. In our experiments we have estimated $\sigma$ using this method.

### B. Factorizing the Matrix with Extracted Item Features

The previous section discusses how one might use the kernel method to extract the item features. This results in a new $n \times k$ matrix $V_0$. Each row of $V_0$ contains a $k$-dimensional representation of an item. Unlike traditional matrix factorization, this representation was approximated using kernel PCA. The next step is to factorize the ratings matrix $R$ as $R \approx UV_0^T$, where $V_0$ is assumed to be fixed, and only the matrix $U$ needs to be learned. It is assumed that the $(i, j)$th entry of $V_0$ is $v_{ij}^0$. Note that this is a simplified form of matrix factorization, which can reduce the number of parameters to be learned. The basic idea here is that the new feature space found by the kernel method is more robust than the one found by traditional matrix factorization because of its ability to adjust to nonlinear manifolds. Therefore, the factorization is performed after fixing the item factors, so that the user factors are also defined in the same space. The prediction of the rating $r_{ij}$ may be performed as $\hat{r}_{ij} = \sum_{s=1}^k u_{is} v_{js}^0$. We would like to minimize the sum of the squared error $e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2$ over all the *observed* entries in the ratings matrix.

As in traditional matrix factorization, a gradient-descent method is used to optimize the error over the observed entries.

Recall that the indices of the observed entries are denoted by $E$. The corresponding optimization model for the modified problem is constructed over the observed entries in $E$:

$$\text{Minimize } J = \frac{1}{2}\sum_{(i,j)\in E}\left(r_{ij} - \sum_{s=1}^{k} u_{is}\cdot v_{js}^0\right)^2 + \frac{\lambda_f}{2}\sum_{i,s} u_{is}^2.$$

Note that in this case, one only needs to solve for the optimization variables $u_{is}$. This can be achieved with the use of gradient-descent methods. Therefore, we can determine the partial derivative of $J$ with respect to $u_{is}$ as follows:

$$\frac{\partial J}{\partial u_{is}} = -\sum_{j:(i,j)\in E} e_{ij}v_{js}^0 + \lambda_f u_{is}. \tag{7}$$

One can then update $u_{is}$ for each $(i,s)$ using a standard gradient-descent update as follows:

$$u_{is} \leftarrow u_{is} - \alpha_f \frac{\partial J}{\partial u_{is}}. \tag{8}$$

It is common to use these updates as a part of the stochastic gradient descent (SGD) optimization. In stochastic gradient-descent, one iterates through the observed entries $(i,j)\in E$ in random order, and makes the following updates for each $s\in\{1,\ldots,k\}$:

$$u_{is} \leftarrow u_{is}(1-\alpha_f\cdot\lambda_f) + \alpha_f e_{ij}v_{js}^0. \tag{9}$$

As in the case of gradient descent, the updates can be executed to convergence, although executing a fixed number of sweeps through the data set works better in practice.

### C. Application to Biased Matrix Factorization

The model discussed in the previous section uses the bias variables for feature extraction, it does not, however, incorporate the bias variables in the second stage of factorization. It has been shown that incorporating bias often improves the performance of a factorization model. Such models are known as Biased Matrix Factorization (BMF) models. We use the kernel features as the item features in order to kernelize the BMF model. Therefore, the final prediction rule for our proposed Kernel Biased Matrix Factorization (K-BMF) model is as follows:

$$\hat{r}_{ij} = b_i^{user} + b_j^{item} + \sum_{s=1}^{k} u_{is}v_{js}^0. \tag{10}$$

Note the use of the kernel features in the aforementioned prediction rule. We generalize the steps used in biased matrix factorization [1] with this kernelized prediction. The gradient descent steps again become much simpler for the K-BMF model because the item factors are no longer optimization variables. Overall, the stochastic gradient descent update steps for the above model can be summarized as follows:

$$\begin{aligned} b_i^{user} &\leftarrow b_i^{user} + \alpha_b(e_{ij} - \lambda_b b_i^{user}) \\ b_i^{item} &\leftarrow b_i^{item} + \alpha_b(e_{ij} - \lambda_b b_i^{item}) \\ u_{is} &\leftarrow u_{is} + \alpha_f(e_{ij}v_{js}^0 - \lambda_f u_{is}). \end{aligned}$$

Here $e_{ij} = r_{ij} - \hat{r}_{ij}$ is the error between the ratings predicted using Eq. 10 and observed ratings. Observe that we have used two different learning rates ($\alpha_b$, $\alpha_f$) and regularization parameters ($\lambda_b$, $\lambda_f$). This is because we follow a common practice of using different learning rates and regularization terms for the bias and factorization component of the model.

### D. Kernelizing Implicit Feedback Factorization

The basic idea behind implicit feedback is that the *act of rating an item* has tremendous predictive value in its own right, irrespective of the actual value of the rating. Therefore, in the simplest case, explicit feedback can be converted into implicit feedback as a binary matrix $F$ in which rated items take on the value of 1 and unrated items take on the value of 0. The rows of this matrix can then be normalized to unit norm. Therefore, each non-zero (i.e., rated) entry in the row of $F$ for user $u$ is given by $1/\sqrt{|N(u)|}$, where $N(u)$ is the set of items rated by user $u$. A popular implicit feedback factorization model is the SVD++ model. The SVD++ model shows that one can use the derived implicit feedback matrix $F$ to further improve the predictions.

The SVD++ model extends biased matrix factorization by learning implicit feedback factors for the users. Therefore a prediction for item $i$ and user $j$ not only depends on the ratings of other items similar to $i$ that $j$ rates, but also on how much predictive power is contained in the act of rating these items by any user. The original SVD++ model learns the user, item, and implicit feedback factors from the ratings matrix $R$ as follows:

$$R = \text{Bias Adjustment} + (U + FY)V^T. \tag{11}$$

Here, $F$ is the aforementioned $m\times n$ implicit feedback matrix with unit-norm rows, and $Y$ is an $n\times n$ matrix of item-to-item implicit feedback affinities that need to be learned along with $U$ and $V$. We modify this model and use the fixed kernel latent item factors $V_0$ (derived in section III-A) instead of learning the item factors $V$. The prediction rule for the modified SVD++ model is given as follows:

$$\hat{r}_{ij} = b_i^{user} + b_j^{item} + \sum_{s=1}^{k} v_{js}^0\left(u_{is} + \frac{\sum_{t\in N(u)} y_{jt}}{\sqrt{|N(u)|}}\right). \tag{12}$$

The variables $y_{jt}$ represent the item-to-item implicit feedback affinities. The corresponding optimization model is constructed using only the observed entries as follows:

$$\begin{aligned} \text{Min. } J =& \frac{1}{2}\sum_{(i,j)\in E} e_{ij}^2 + \frac{\lambda_b}{2}\left(\sum_i (b_i^{user})^2 + \sum_j (b_j^{item})^2\right) \\ &+ \frac{\lambda_f}{2}\left(\sum_{i,s} u_{is}^2 + \sum_{j,t} y_{jt}^2\right). \end{aligned}$$

Here, $e_{ij} = r_{ij} - \hat{r}_{ij}$ is the error between the ratings predicted using Eq. 12 and observed ratings. The above optimization model is partially differentiated with respect to the bias ($b_i^{user}, b_j^{item}$), factorization ($u_{is}$), and implicit feedback ($y_{jt}$)

terms. The gradient steps for $b_i^{user}$, $b_i^{item}$, and $u_{is}$ are identical to the K-BMF model, while $y_{jt}$ is updated as follows:

$$\forall t \in N(u) : y_{jt} \leftarrow y_{jt} + \alpha_f \left( e_{ij} \frac{v_{js}^0}{\sqrt{|N(u)|}} - \lambda_f y_{jt} \right).$$

These gradient descent iterations are similar to the original SVD++ model [2]. The only difference is that the new iterations treat all the $v_{js}^0$ as constants, while the other parameters are treated as optimization variables. We denote the model given in Eq. 12 as K-SVD++.

## IV. Experimental Results

In this section we provide an extensive evaluation of the methods proposed in this paper. The summary statistics of all data sets used for the experiments are shown in Table I.

Table I
Summary of the Data Sets.

| Data set | Users | Items | Ratings | Density (%) |
|---|---|---|---|---|
| FilmTrust | 1,508 | 2,071 | 35,497 | 1.13 |
| ML100K | 943 | 1,682 | 100,000 | 6.30 |
| Ciao | 7,257 | 10,000 | 141,984 | 0.20 |
| Epinions | 21,427 | 10,000 | 385,358 | 0.18 |
| Jester | 63,978 | 150 | 1,761,439 | 18.35 |

ML100K[1] and FilmTrust[2] are movie recommendation data sets, while the Jester[3] data set is a joke recommendation data set. The Ciao and Epinions data sets[4] are related to music and product ratings respectively. For the Ciao and Epinions data sets we use only the top 10,000 most rated items. We use root mean-squared error (RMSE) as an accuracy measure for all the methods. Lower values of RMSE are desirable. For consistency, we compared the kernelized version of a method with its base method, and we use comparable parameter settings for both. The data set is split into a 75%-25% train and test split. Average test RMSE on 10 such independent splits is reported.

We begin by comparing Biased Matrix Factorization (BMF) and SVD++ with the proposed Kernel-Based Biased Matrix Factorization (K-BMF) and Kernel-Based SVD++ (K-SVD++). We use comparable parameter settings for all of these approaches. We set the factorization rank $k = 10$, and the learning rates for the bias and factorization components as $\alpha_b = 0.01$ and $\alpha_f = 0.01$. The results are reported for both high and low levels of regularization. One reason for doing so is to show that the approach can perform well in settings corresponding to varying levels of overfitting. For the low regularization case, we used $\lambda_b = 0.005$ and $\lambda_f = 0.015$, while for high regularization we set $\lambda_b = 0.05$ and $\lambda_f = 0.15$. For all the algorithms, we estimated the model parameters using Stochastic Gradient Descent (SGD), which is executed for 10 full sweeps over the entire set of ratings.

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

[2]http://www.librec.net/datasets.html

[3]http://www.ieor.berkeley.edu/~goldberg/jester-data/

[4]http://www.public.asu.edu/~jtang20/datasetcode/truststudy.htm

### A. Accuracy Results

The accuracy of various factorization methods over all data sets are reported in Tables II and III for low and high levels of regularization, respectively. The methods that use kernel features are clearly superior and are relatively insensitive to the level of regularization. Out of five data sets, the kernel-based methods convincingly win on four datasets. Higher regularization helps BMF and SVD++ because they learn two sets of factor matrices at a given time, but even with higher regularization they are unable to perform better than the proposed K-BMF and K-SVD++ methods. This demonstrates the effectiveness of using kernel features for the collaborative filtering task.

The maximum improvement of K-BMF over BMF is observed for the Jester data set. This improvement is about 17% percent and 13% percent for low and high regularization, respectively. Similarly, when K-SVD++ is compared with SVD++ the maximum improvement of 20% percent and 12% percent can be observed for the low and high regularization cases for the Jester data set. Note that the Jester data set has a relatively large number of specified ratings in relation to other data sets, and it has only 150 items, compared to thousands of items in other data sets. These conditions also enable accurate estimation of the kernel matrix and the kernel features. The improvements in other data sets were smaller in comparison, but still quite large compared to what is typical[5] in the collaborative filtering domain. Overall, the methods using kernel features exhibit significant accuracy enhancements as compared to their counterparts that do not use these features. More importantly, the *consistency* of this generic methodology over different data sets, algorithms, and regularization conditions is noteworthy. High regularization improves the accuracy of the baseline factorization methods drastically. Even so, the kernel-based method is almost always able to outperform the baseline methods.

### B. Effect of Factorization Rank

In this section we will demonstrate the impact of the factorization rank on accuracy. These set of experiments are performed for both the low and high regularization scenarios. Due to space constraints the results are only shown for the ML100K and Jester data sets. For these set of experiments we set the learning rates at $\alpha_b = 0.01$ and $\alpha_f = 0.01$. The effect of the factorization rank for low and high regularization are shown in Figure 2 and Figure 3 respectively. The regularizers for the baseline and factor variables for the case of high regularization are set at $\lambda_b = 0.05$ and $\lambda_f = 0.15$, while for the case of low regularization, their values are $\lambda_b = 0.005$ and $\lambda_f = 0.015$.

It is clear that the baseline factorization methods are not robust to increase in the factorization rank for both the low and high regularization scenarios. In the case of low regularization (refer Figure 2), the RMSE on the test set for the BMF method

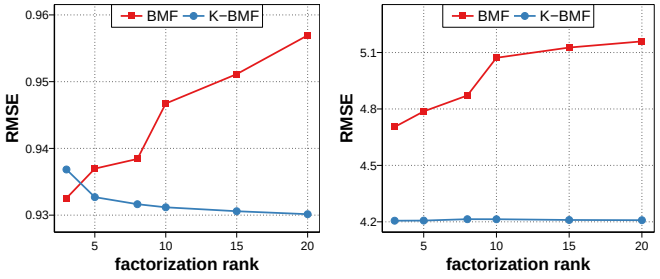
[5]In the Netflix Prize challenge, it took multiple teams several years to improve accuracy by 10% over a relatively trivial baseline predictor.

Table II
**LOW REGULARIZATION:** ACCURACY COMPARISON FOR LOW REGULARIZATION ($\lambda_b = 0.005, \lambda_f = 0.015$).

| | Data set | | | | |
|---|---|---|---|---|---|
| Methods | FILMTRUST | ML100K | CIAO | EPINIONS | JESTER |
| BMF | 0.8120 | 0.9467 | 0.9835 | 1.0704 | 5.0720 |
| K-BMF | **0.7988** | **0.9312** | **0.9663** | **1.0462** | **4.2139** |
| SVD++ | 0.8133 | 0.9352 | 1.0001 | 1.0827 | 5.2920 |
| K-SVD++ | **0.7993** | **0.9305** | **0.9671** | **1.0465** | **4.2111** |

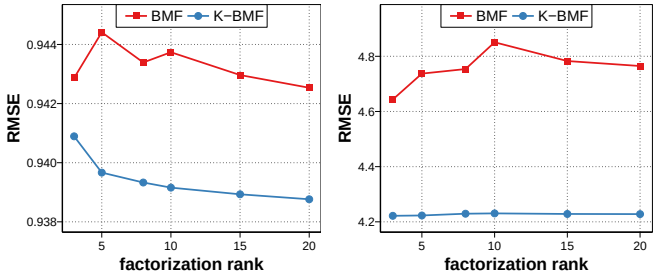
Table III
**HIGH REGULARIZATION:** ACCURACY COMPARISON FOR HIGH REGULARIZATION ($\lambda_b = 0.05, \lambda_f = 0.15$).

| | Data set | | | | |
|---|---|---|---|---|---|
| Methods | FILMTRUST | ML100K | CIAO | EPINIONS | JESTER |
| BMF | 0.8007 | 0.9437 | 0.9661 | **1.0439** | 4.8508 |
| K-BMF | **0.7982** | **0.9391** | **0.9641** | 1.0441 | **4.2306** |
| SVD++ | 0.8007 | 0.9410 | 0.9662 | **1.0437** | 4.8160 |
| K-SVD++ | **0.7984** | **0.9391** | **0.9642** | 1.0441 | **4.2301** |




(a) ML100K (b) JESTER

Figure 2. Low Regularization ($\lambda_b = 0.005, \lambda_f = 0.015$): Effect of the number of factors on the accuracy of factorization methods.




(a) ML100K (b) JESTER

Figure 3. High Regularization ($\lambda_b = 0.05, \lambda_f = 0.15$): Effect of the number of factors on the accuracy of factorization methods.

actually increases as the number of factors increase. Since factorization methods estimate all the parameters from the data in a single step they are more prone to overfitting. On the other hand, since the kernel-based methods first derive relevant features and then use them to learn a smaller set of parameters, they are less prone to overfitting in addition to the fact that the derived features are superior as compared to the set of item factors learned from the data using SGD. When the regularization is increased (refer Figure 3) BMF shows limited improvement or the accuracy remains stable. Notice that even though the performance of BMF improves with high regularization, it is unable to obtain better RMSE as compared to K-BMF.

## V. CONCLUSIONS

In this paper, we showed a general technique to enhance existing collaborative filtering methods with the use of kernel features. As shown by our experimental results, this approach can be used to improve the performance of traditional matrix factorization, biased matrix factorization and SVD++ on a consistent basis. This is a significant advantage because it can improve many off-the-shelf collaborative filtering methods with modest modifications. Our experimental results show that we are able to obtain consistent improvements with this approach in a variety of models. In our future work, we will address the computational challenges of incorporating such kernel-centric methods into user-based models.

## REFERENCES

[1] C. Aggarwal. Recommender Systems, *Springer*, 2016.
[2] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. *KDD*, 2008.
[3] X. Liu, C. Aggarwal, Y. Li, X. Kong, X. Sun, and S. Sathe. Kernelized matrix factorization for collaborative filtering. *SIAM Conference on Data Mining*, pp. 319–416, 2016.