Incremental Kernel Mapping Algorithms for Scalable Recommender Systems

Mustansar Ali Ghazanfar
School of Electronics and Computer Science, University of Southampton, Highfield Campus, SO17 1BJ, U.K.
Email: mag208r@ecs.soton.ac.uk

Sandor Szedmak
Intelligent and Interactive Systems, University of Innsbruck, 6020 Innsbruck, Austria
Email: sandor.szedmak@uibk.ac.at

Adam Prugel-Bennett
School of Electronics and Computer Science, University of Southampton, Highfield Campus, SO17 1BJ, U.K.
Email: apb@ecs.soton.ac.uk

***Abstract*—Recommender systems apply machine learning techniques for filtering unseen information and can predict whether a user would like a given item. Kernel Mapping Recommender (KMR) system algorithms have been proposed, which offer state-of-the-art performance. One potential drawback of the KMR algorithms is that the training is done in one step and hence they cannot accommodate the incremental update with the arrival of new data making them unsuitable for the dynamic environments. From this line of research, we propose a new heuristic, namely KMR^{incr}, which can build the model incrementally without retraining the whole model from scratch when new data (item or user) are added to the recommender system dataset. Furthermore, we proposed a novel perceptron-type algorithm, namely $KMR^{percept}$, which is a fast incremental algorithm for building the model that maintains a good level of accuracy and scales well with the data. We show empirically over two datasets that the proposed algorithms give quite accurate results while providing significant computation savings.**

***Index Terms*—Recommender Systems, Incremental Algorithm, Maximum Margin, Kernel, Perceptron.**

I. Introduction

A. Recommender Systems Background

There has been an exponential increase in the volume of available digital information, electronic sources, and online services in recent years. This information overload has created a potential problem, which is how to filter and efficiently deliver relevant information to a user. These problems highlight a need for information extraction systems that can filter unseen information and can predict whether a user would like a given item. Such systems are called recommender systems, and they mitigate the aforementioned problem to a great extent.

A recommender system consists of two basic entities: users and items, where users provide their opinions (ratings) about items. We denote these users by $\mathcal{U} = \{u_1, u_2, \cdots, u_M\}$, where the number of people using the system is $|\mathcal{U}| = M$ and denote the set of items being recommended by $\mathcal{I} = \{\ i_1, i_2, \cdots, i_N\ \}$, with $|\mathcal{I}| = N$. The users will have given ratings of some, but not all of the items. We denote these ratings by $(r_{iu}|(i,u) \in \mathcal{D})$, where $\mathcal{D} \subset \mathcal{U} \times \mathcal{I}$ denotes the set of user-item pairs that have been rated. We denote the total number of ratings made by $|\mathcal{D}| = T$. Typically each user rates only a small number of the possible items so that $|\mathcal{D}| = T \ll |\mathcal{U} \times \mathcal{I}| = M \times N$. It is not unusual in practical systems to have $T/(M \times N) \approxeq 0.01$. We can denote the ratings made by the users by a $M \times N$ rating matrix R with elements $r_{i,j}$. We denote the items for which there are ratings by user u as $\mathcal{D}_u$ and the users who have rated an item i by $\mathcal{D}_i$. The task is to create a recommender algorithm that predicts an unseen rating r_{iu}, i.e. for $(i,u) \notin \mathcal{D}$.

B. Problem Statement

The Kernel Mapping Recommender (KMR) system algorithms have been proposed [1], which claim to offer the state-of-the-art performance. A potential drawback of the KMR algorithms is that the training occurs in one step. For practical recommender systems this is a significant problem, as with the incremental and gradual arrival of the new data, it is desirable that the updates are performed on such a data. It is unrealistic to recompute the model from scratch, based on these updates, due to the tremendous cost related to computation time and storage capacity. Against this background, we propose a perceptron-like algorithm that we call $KMR^{percept}$, which can incrementally build the model by sequentially processing the data points one at a time as they arrive. Furthermore, we propose a heuristic method that we call KMR_{Inc}, which can be used to update the model effectively on the arrival of new data.

Both of the proposed algorithms overcome the *accuracy* and *scalability* problems [2], [3] associated with a recommender system. Furthermore, the proposed algorithm, $KMR^{percept}$, overcomes the *stability vs. plasticity problem* (sometimes referred to as *user-interest drifting problem*) [4]—once a detailed user's profile has been built, then it becomes very difficult for these systems to change this profile. For example, in a movie recommender system, if a user was interested in action movies last year, but their taste then changed to romantic movies, then they would not receive useful recommendation. This is because, up to this point, their profile has been heavily shaped by action movies. Our algorithm $KMR^{percept}$ can solve this problem by giving more weight to the current (or recent) ratings.

The rest of the paper has been organised as follows. Section 2 describes the background concepts related to the kernel mapping recommender. Section 3 outlines the proposed incremental algorithms namely $KMR^{percept}$ and KMR^{incr}. Section 4 describes the data set and metrics used in this work.

Section 5 presents results comparing the performance of the proposed algorithms with the baseline ones, followed by the discussion, and finally section 6 concludes the work.

II. Background: Kernel Mapping Recommender (KMR) System Algorithm

In [1] the authors proposed Kernel Mapping Recommender (KMR) algorithms for solving the recommender system problem. We first describe how an item-based recommender system can be built and then show how the described approach can be generalised to a user-based recommender. To perform the recommendation task we consider building the additive model for the *residual ratings* as shown below:

$$\hat{r}_{iu} = r_{iu} - \bar{r}_i - \bar{r}_u + \bar{r},$$

where $\bar{r}_i$, $\bar{r}_u$ and $\bar{r}$ are respectively the mean rating for the item, of the user, and the overall mean

$$\bar{r}_i = \frac{1}{|\mathcal{D}_i|}\sum_{u\in\mathcal{D}_i} r_{iu}, \quad \bar{r}_u = \frac{1}{|\mathcal{D}_u|}\sum_{i\in\mathcal{D}_u} r_{iu}, \quad \bar{r} = \frac{1}{|\mathcal{D}|}\sum_{u\in\mathcal{D}} r_{iu}.$$

We assume that we have some information about the items we denote by $\boldsymbol{q}_i$. This may, for example, be the set of ratings r_{iu} for $u \in \mathcal{D}_i$, or it could be text describing the item i. We map the information to some vector $\boldsymbol{\phi}(\boldsymbol{q}_i)$ in some extended feature (Hilbert) space. Similarly we map the rating residues, $\hat{r}_{iu}$, to some Hilbert space $\boldsymbol{\psi}(\hat{r}_{iu})$. In this paper, we consider these objects to lie in a function space $\boldsymbol{\psi}(\hat{r}_{iu}) = \mathcal{N}(x|\hat{r}_{iu}, \sigma)$ where $\mathcal{N}(x|\mu,\sigma)$ is the space of density functions of the normal distribution with mean μ and variance σ^2. The motivation of this choice is to model possible errors in the rating either due to the discretisation of the rating scale or the subjective variability in assigning a rating.

The method seeks a linear mapping between these two spaces which can be used for making predictions. More specifically, we look for a linear mapping $\mathcal{W}_u$ from the space of $\boldsymbol{\phi}$ vectors to the space of $\boldsymbol{\psi}$ vectors, such that the inner produce satisfies the inequality

$$\langle \boldsymbol{\psi}(\hat{r}_{iu}), \mathcal{W}_u\boldsymbol{\phi}(\boldsymbol{q}_i)\rangle \geq 1 - \zeta_i$$

where $\zeta_i \geq 0$ is a slack variable. We then seek to minimise the Frobinius norm of $\mathcal{W}_u$ and the slack variables ζ_i. More specifically we minimise

$$\frac{1}{2}\sum_{u\in\mathcal{U}} \|\mathcal{W}_u\|^2 + C\sum_{i\in\mathcal{I}} \zeta_i \tag{1}$$

subject to the above constraints. Note that minimisation will be achieved when the vectors $\mathcal{W}_u\boldsymbol{\phi}(\boldsymbol{q}_i)$ are as aligned as possible with the vector $\boldsymbol{\psi}(\hat{r}_{iu})$.

To solved this constrained optimisation problem we define the Lagrangian

$$\mathcal{L} = \frac{1}{2}\sum_{u\in\mathcal{U}} \|\mathcal{W}_u\|^2 + C\sum_{i\in\mathcal{I}} \zeta_i - \sum_{(i,u)\in\mathcal{D}} \alpha_{iu}\left(\langle \boldsymbol{\psi}(\hat{r}_{iu}), \mathcal{W}_u\boldsymbol{\phi}(\boldsymbol{q}_i)\rangle - 1 + \zeta_i\right) - \sum_{i\in\mathcal{I}} \lambda_i\zeta_i$$

where $\alpha_{iu} \geq 0$ are Lagrange multipliers to ensure that $\langle \boldsymbol{\psi}(\hat{r}_{iu}), \mathcal{W}_u\boldsymbol{\phi}(\boldsymbol{q}_i)\rangle \geq 1 - \zeta_i$ and $\lambda_i \geq 0$ are Lagrange multipliers to ensure that $\zeta_i \geq 0$. The optimum mapping is found by solving

$$\min_{\{\mathcal{W}_u\},\{\zeta_i\}} \max_{\{\alpha_{iu}\},\{\lambda_i\}} \mathcal{L}$$

subject to the constraints that $\alpha_{iu} \geq 0$ for all $(i,u) \in \mathcal{D}$ and $\lambda_i \geq 0$ for all $i \in \mathcal{I}$.

After solving this problem we find that the initial objective function is maximised when the α_{iu} maximises the function

$$f(\boldsymbol{\alpha}) = -\frac{1}{2}\sum_{u\in\mathcal{U}}\sum_{i,i'\in\mathcal{D}_u} \alpha_{iu}\alpha_{i'u}\langle \boldsymbol{\psi}(\hat{r}_{iu}), \boldsymbol{\psi}(\hat{r}_{i'u})\rangle\langle \boldsymbol{\phi}(\boldsymbol{q}_i), \boldsymbol{\phi}(\boldsymbol{q}_{i'})\rangle + \sum_{(i,u)\in\mathcal{D}} \alpha_{iu}$$

subject to the constraint that $\boldsymbol{\alpha} \in Z(\boldsymbol{\alpha})$ where

$$Z(\boldsymbol{\alpha}) = \left\{ \boldsymbol{\alpha} \middle| \forall u \in \mathcal{U}, \sum_{u\in\mathcal{D}_i} \alpha_{iu} < C \wedge \forall (i,u) \in \mathcal{D}, \alpha_{iu} \geq 0 \right\}.$$

We are now in a position where we can apply the usual kernel trick. Defining the kernel functions

$$K_{\hat{r}}(\hat{r}_{iu}, \hat{r}_{i'u}) = \langle \boldsymbol{\psi}(\hat{r}_{iu}), \boldsymbol{\psi}(\hat{r}_{i'u})\rangle$$
$$K_{\boldsymbol{q}}(\boldsymbol{q}_i, \boldsymbol{q}_{i'}) = \langle \boldsymbol{\phi}(\boldsymbol{q}_i), \boldsymbol{\phi}(\boldsymbol{q}_{i'})\rangle$$

we can write $f(\boldsymbol{\alpha})$ as

$$f(\boldsymbol{\alpha}) = -\frac{1}{2}\sum_{u\in\mathcal{U}}\sum_{i,i'\in\mathcal{D}_u} \alpha_{iu}\alpha_{i'u}K_{\hat{r}}(\hat{r}_{iu}, \hat{r}_{i'u})K_{\boldsymbol{q}}(\boldsymbol{q}_i, \boldsymbol{q}_{i'}) + \sum_{(i,u)\in\mathcal{D}} \alpha_{iu}$$

where we are free to choose any pair of positive definite kernel functions.

For large-scale recommender systems, solving this quadratic programming problem using a general quadratic programming solver would be impractical due to the large number of data points. However, we can find an approximate solution iteratively using the conditional gradient method.

To understand this method it is helpful to write $f(\boldsymbol{\alpha})$ in matrix form

$$f(\boldsymbol{\alpha}) = -\frac{1}{2}\boldsymbol{\alpha}^\mathsf{T}\mathbf{M}\boldsymbol{\alpha} + \boldsymbol{b}^\mathsf{T}\boldsymbol{\alpha}$$

with $\boldsymbol{\alpha} \in Z(\boldsymbol{\alpha})$. We obtain a series of approximations $\boldsymbol{\alpha}_t$ for the optimal parameters starting from some initial guess $\boldsymbol{\alpha}_0 \in Z(\boldsymbol{\alpha})$. At each step we use a linear approximation for $f(\boldsymbol{\alpha})$

$$f(\boldsymbol{\alpha}) \approx \hat{f}_{\boldsymbol{\alpha}_t}(\boldsymbol{\alpha}) = f(\boldsymbol{\alpha}_t) + (\boldsymbol{\alpha} - \boldsymbol{\alpha}_t)\boldsymbol{\nabla} f(\boldsymbol{\alpha}_t)$$

We compute the next approximation using two stages. We first solve the linear programming problem

$$\begin{aligned}\boldsymbol{\alpha}^* &= \operatorname*{argmax}_{\boldsymbol{\alpha}\in Z(\boldsymbol{\alpha})} \hat{f}_{\boldsymbol{\alpha}_t}(\boldsymbol{\alpha}) \\ &= \operatorname*{argmax}_{\boldsymbol{\alpha}\in Z(\boldsymbol{\alpha})} -\boldsymbol{\alpha}^\mathsf{T}(\mathbf{M}\boldsymbol{\alpha}_t - \boldsymbol{b}) + \text{const.}\end{aligned}$$

We then find the new approximation $\boldsymbol{\alpha}_{t+1}$ to be

$$\boldsymbol{\alpha}_{t+1} = \boldsymbol{\alpha}_t + \tau(\boldsymbol{\alpha} - \boldsymbol{\alpha}_t)$$

where we choose τ to be

$$\tau = \underset{\tau}{\operatorname{argmax}}\, f(\boldsymbol{\alpha}_{t+1}) = \frac{(\boldsymbol{b} + \mathbf{M}\boldsymbol{\alpha}_t)^\intercal(\boldsymbol{\alpha}^* - \boldsymbol{\alpha}_t)}{(\boldsymbol{\alpha}^* - \boldsymbol{\alpha}_t)^\intercal \mathbf{M}(\boldsymbol{\alpha}^* - \boldsymbol{\alpha}_t)}.$$

This guarantees that each step does not increase the objective function.

To make a prediction for the rating r_{iu} where $(i,u) \notin \mathcal{D}$ we make a prediction for the residue $\hat{r}_{iu} = r_{iu} - \bar{r}_i - \bar{r}_u + \bar{r}$ using the function

$$\begin{aligned} p_{iu}(\hat{r}) &= \langle \psi(\hat{r}), \mathcal{W}_u \phi(\boldsymbol{q}_i) \rangle \\ &= \sum_{i' \in \mathcal{D}_u} \alpha_{iu}\alpha_{i'u} K_{\hat{r}}(\hat{r}, \hat{r}_{i'u}) K_{\boldsymbol{q}}(\boldsymbol{q}_i, \boldsymbol{q}_{i'}). \end{aligned}$$

We have a choice in how to obtain a single prediction from this function. We can choose the 'max'

$$\hat{r}_{iu} = \underset{\hat{r}}{\operatorname{argmax}}\, p(\hat{r}).$$

To perform a user-based recommendation we use information $\boldsymbol{q}_u$ about users u and try to find a linear mapping $\mathcal{W}_i$ to align some extended feature vectors $\boldsymbol{\phi}(\boldsymbol{q}_u)$ to the residue vector $\boldsymbol{\psi}(\hat{r}_{iu})$. The derivation is identical to that for the item-based recommender when we interchange the subscripts i and u.

III. INCREMENTALLY BUILDING THE MODEL

In this section, we describe the proposed algorithms to build the model on the arrival of new data. In the following the base dataset, denoted by $\mathcal{D}^{base}$, represents the dataset used to train the initial model and the resulting model is called the base model. Similarly, the dataset added afterwards, denoted by $\mathcal{D}^{new}$, represents the new dataset and the resulting model is called the updated model.

A. KMR^{incr}

The proposed algorithm is outlined in Algorithm 1. From step 2 to 3, we initialise the model parameters: items vector, users vector, total average, design variable, and design variable's initialization parameter. We then build the base model with the dataset defined as the base model dataset (i.e. $\mathcal{D}^{base}$). Let $\tilde{\mathcal{I}}$, $\tilde{\mathcal{U}}$, $\tilde{E}$, $\tilde{E}_i$, $\tilde{E}_u$, $\tilde{\alpha}$ represent the base model items, users, total averages, item averages, user averages, and the design variables respectively. In step 5, we compute the mean of the design variables ($\tilde{\alpha}$) computed while building the base model. We update the model adding the new dataset (i.e. $\mathcal{D}^{new}$) to the existing dataset and initialise the new model parameters by the base model parameters.

In the solver procedure, from steps 9 to 11, we read the new data. From steps 12 to 17, we initialise the design variables which is different for the base model dataset (i.e. $\mathcal{D}^{base}$) and the dataset added afterwards (i.e. $\mathcal{D}^{new}$). Formally, the

Algorithm 1 :KMR^{incr}, Build and update the model

1: **procedure** BUILDMODEL($\mathcal{D}^{base}, \mathcal{D}^{new}$)
2: $\mathcal{I} = \emptyset$; $\mathcal{U} = \emptyset$;
3: $E_i = 0$, $E_u = 0$, $E = 0$, $\alpha = 0$ $E_\alpha = 0$
4: ## Build the Base Model
$\tilde{\mathcal{I}}, \tilde{\mathcal{U}}, \tilde{E}, \tilde{E}_i, \tilde{E}_u, \tilde{\alpha}$) = SOLVER ($\mathcal{I}, \mathcal{U}, E, E_i, E_u, E_\alpha, \alpha, \mathcal{D}^{base}$)
5: $E_{\tilde{\alpha}} = \dfrac{\sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} \tilde{\alpha}_{iu}}{\text{card}(\mathcal{U})\text{card}(\mathcal{I})}$
6: ## Update the base model
$\mathcal{I}, \mathcal{U}, E, E_i, E_u, \alpha$) = SOLVER ($\tilde{\mathcal{I}}, \tilde{\mathcal{U}}, \tilde{E}, \tilde{E}_i, \tilde{E}_u, \tilde{E}_\alpha, \tilde{\alpha}, \mathcal{D}^{new} \cup \mathcal{D}^{base}$)
7: **end procedure**

Solve the optimisation problem and find the design variables
8: **procedure** SOLVER($\mathcal{I}, \mathcal{U}, E(t-1), E_i(t-1), E_u(t-1), E_\alpha, \alpha, \mathcal{D}$)
9: **for all** $t \in \{1, \cdots, \mathcal{D}\}$ **do**
10: read ($i(t)$,$u(t)$,$r_{iu}(t)$)
11: $i = i(t)$;$u = u(t)$;$r_{iu} = r_{iu}(t)$
12: ## Initialization for new, earlier unseen items and users
13: **if** $i \notin \mathcal{I}$ || $u \notin \mathcal{U}$ **then**
14: $\alpha_{iu}(t) = E_\alpha$
15: **else**
16: $\alpha_{iu}(t) = \tilde{\alpha}_u(t)$
17: **end if**
18: **if** $i \notin \mathcal{I}$ **then**
19: $\mathcal{I} = \mathcal{I} \cup \{i\}$; $\mathcal{U}_i = \emptyset$
20: $E_i(t-1) = 0$
21: **end if**
22: **if** $u \notin \mathcal{U}$ **then**
23: $\mathcal{U} = \mathcal{U} \cup \{u\}$; $\mathcal{I}_u = \emptyset$
24: $E_u(t-1) = 0$
25: $U_i = \mathcal{U}_i \cup \{u\}$
26: $\mathcal{I}_u = \mathcal{I}_u \cup \{i\}$
27: **end if**
28: ## Update average values
29: $E(t) = \dfrac{(t-1)E(t-1) + r_{iu}(t)}{t}$
30: $E_u(t) = \dfrac{(\text{card}(\mathcal{I}_u) - 1)E_u(t-1) + r_{iu}(t)}{\text{card}(\mathcal{I}_u)}$
31: $E_i(t) = \dfrac{(\text{card}(\mathcal{U}_i) - 1)E_i(t-1) + r_{iu}(t)}{\text{card}(\mathcal{U}_i)}$
32: ## Compute residual ranks and inner products
33: $\hat{r}_{iu} = \hat{r}_{iu}(t) = r_{iu}(t) - E_i(t) - E_u(t) + E(t)$
34: $\mathbf{q}_i(t) = (\hat{r}_{iu} \mid u \in \mathcal{U}_i)$
35: ## We use the notation in the sequel for any n, u, t
36: $\gamma_{nu}(t) \overset{\text{def}}{=} \langle \psi(\hat{r}_{iu}(t)), \psi(\hat{r}_{nu}(t)) \rangle \langle \phi(\mathbf{q}_i(t)), \phi(\mathbf{q}_n(t)) \rangle$
37: **end for**
38: ## Solve the optimisation problem given $\mathcal{I}, \mathcal{U}, E(t), E_u(t), E_i(t), \alpha$
39: **return** ($\mathcal{I}, \mathcal{U}, E(t), E_u(t), E_i(t), \gamma_{nu}(t), \alpha$)
40: **end procedure**

initialisation of the α parameter for two different type of dataset is as follows:

$$\alpha_{iu} = \begin{cases} \frac{\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}}\tilde{\alpha}_{iu}}{\text{card}(\mathcal{I})\text{card}(\mathcal{I})} : & \text{if } (i \notin \tilde{\mathcal{I}} \text{ OR } u \notin \tilde{\mathcal{U}}) \\ \tilde{\alpha}_{iu} : & \text{otherwise.} \end{cases} \quad (2)$$

From steps 18 to 27, we keep track of the users and items entering the system by updating the users' and items' arrays. From steps 28 to 31, we update the total, user, and item averages. From steps 32 to 36, we compute the residual ranks, feature vectors, and define the kernel functions as described in section II. At this stage, we compute (in the case of base model-building phase) or update (in the case of update model-building phase) the kernel matrix . Afterwards, we can find the solution (i.e. find the design variables) by feeding the above found parameters to the optimiser, as shown in step 38.

B. Perceptron Algorithm ($KMR^{percept}$)

The online implementation of the recommender system is realised via a perceptron type algorithm. The problem behind the perceptron algorithm is derived from optimisation problem of (1) by ignoring the regularisation term and only minimising the sum of the slack variables, i.e. $\sum_{i\in\mathcal{I}}\zeta_i$ which measures the value of the overall loss. The implemented version of perceptron algorithm follows the dual perceptron schema where the knowledge of the corresponding kernels is required only.

The proposed perceptron like algorithm $KMR^{percept}$ has been outlined in Algorithm 2. From steps 2 to 4, we initialise the model parameters: update counter, items' vector, users' vector, total average, step size, discount factor, and discount parameter. From steps 5 to 7, we read the new arriving data. From steps 8 to 21, we initialise the design variable of the rating made by the user u on item i to zero if either the user or item is new; initialise the item slack variable to zero if the arriving item has not been rated by anyone in the system; and keep track of how many users and items have entered the system by updating the user and item arrays. From steps 22 to 25, we update the total, user, and item averages. From steps 26 to 28 we compute the residual ranks and feature vectors. Then, from steps 29 to 30, we define the kernels as described in section II. The part of the algorithm from steps 32 to 42 implements an incremental subgradient descent step to minimize a problem similar to the one described in section 2. The difference is that the online algorithm minimizes only the sum of the slack variables in (1) and omits the regularisation term. The condition in line 32 selects the cases when the subgradient is not equal to zero.

To make the algorithm more robust, a discounting factor based averaging is carried out when the design variables of the underlying optimisation problem are updated. Refer to the steps in the Algorithm 2 after the comment line "## Discounted update of the variables", where β is the discounting factor and it is chosen from the open interval $(0, 1)$. The discounting can reduce the effect of the earlier observations on the most recent estimation of the variables, since at the

Algorithm 2 : $KMR^{percept}$, sequentially process the data and build the model

1: **procedure** BUILDMODEL($\mathcal{D}$)
2: ##initialise model parameters
3: $k = 0$; $\mathcal{I} = \emptyset$; $\mathcal{U} = \emptyset$; $E(t) = 0$; $s > 0$; ##step size
4: $0 < \beta < 1$; $\beta_k = 1$ ##discount factor and discount initialisation
5: **for all** $t \in \{1, \cdots, \mathcal{D}\}$ **do**
6: read $(i(t),u(t),r_{iu}(t))$
7: $i = i(t); u = u(t); r_{iu} = r_{iu}(t)$
initialisation for new, earlier unseen items and users
8: **if** $i \notin \mathcal{I}$ || $u \notin \mathcal{U}$ **then**
9: $\alpha_{iu}(t) = 0$
10: **end if**
11: **if** $i \notin \mathcal{I}$ **then**
12: $\mathcal{I} = \mathcal{I} \cup \{i\}$; $\mathcal{U}_i = \emptyset$
13: $E_i(t-1) = 0$
14: $\zeta_i(t) = 0$
15: **end if**
16: **if** $u \notin \mathcal{U}$ **then**
17: $\mathcal{U} = \mathcal{U} \cup \{u\}$; $\mathcal{I}_u = \emptyset$
18: $E_u(t-1) = 0$
19: $U_i = \mathcal{U}_i \cup \{u\}$
20: $\mathcal{I}_u = \mathcal{I}_u \cup \{i\}$
21: **end if**
22: ## Update average values
23: $E(t) = \dfrac{(t-1)E(t-1) + r_{iu}(t)}{t}$
24: $E_u(t) = \dfrac{(\text{card}(\mathcal{I}_u) - 1)E_u(t-1) + r_{iu}(t)}{\text{card}(\mathcal{I}_u)}$
25: $E_i(t) = \dfrac{(\text{card}(\mathcal{U}_i) - 1)E_i(t-1) + r_{iu}(t)}{\text{card}(\mathcal{U}_i)}$
26: ## Compute residual ranks
27: $\hat{r}_{iu} = \hat{r}_{iu}(t) = r_{iu}(t) - E_i(t) - E_u(t) + E(t)$
28: $\mathbf{q}_i(t) = (\hat{r}_{iu} \ | u \in \mathcal{U}_i)$
29: ## We use the notation in the sequel for any n, u, t
30: $\gamma_{nu}(t) \stackrel{\text{def}}{=} \langle \psi(\hat{r}_{iu}(t)), \psi(\hat{r}_{nu}(t)) \rangle \langle \phi(\mathbf{q}_i(t)), \phi(\mathbf{q}_n(t)) \rangle$
31: ## Test the constraint belonging to $(i(t), u(t))$
32: **if** $\sum_{n\in\mathcal{I}_u} \alpha_{nu}(t)\gamma_{nu}(t) < 1 - \zeta_i(t)$ **then**
33: ## Discounted update of the variables
34: $\beta_{k+1} = 1 + \beta\beta_k$
35: $k = k + 1$; ## update counter
36: **for all** $n \in \mathcal{I}_u$ **do**
37: $\alpha_{nu}(t+1) = \frac{1}{\beta_k}[\beta\alpha_{nu}(t) + s\gamma_{nu}(t)]$
38: **end for**
39: **if** $\zeta_i(t) > 1 - \sum_{n\in\mathcal{I}_u} \alpha_{nu}(t+1)\gamma_{nu}(t)$ **then**
40: $\zeta_i(t+1) = 1 - \sum_{n\in\mathcal{I}_u} \alpha_{nu}(t+1)\gamma_{nu}(t)$
41: **end if**
42: **end if**
43: **end for**
44: **return** ($\mathcal{I}$, $\mathcal{U}$, $E(t)$, $E_u(t)$, $E_i(t)$, α)
45: **end procedure**

Table I
CHARACTERISTICS OF THE DATASETS USED IN THIS WORK. THE FILMTRUST AND MOVIELENS DATASETS HAVE BEEN SHOWN BY FT AND SML RESPECTIVELY. AVERAGE RATING REPRESENTS THE AVERAGE RATING GIVEN BY ALL USERS IN THE DATASET.

Characteristics	Dataset	
	(FT)	(SML)
Number of users	1 016	943
Number of movies	314	16 82
Number of ratings	25 730	100 000
Rating scale	1.0-10.0	1-5
Sparsity	0.988	0.934
Max number of ratings given by a user	244	737
Max number of ratings given to a movie	880	583
Average rating	7.607	3.529

beginning of the algorithm the values of the averages can have high variance caused by the small sample used to estimate them.

The variable β_k is used to normalise the accumulated values of the discounted variables. The subscript k refers to the number of updates of the design variable in the algorithm presented in Algorithm 2. This normalization gives a convex combination of values of all updates of variables, where the weights diminish more if the update happened earlier. The current values of the β_k are computed by a recursive formula based on the Horner schema in the algorithm

$$\beta_0 = 1,\ \beta_k = 1 + \beta\beta_{k-1} \text{ for any index } \quad k > 0, \tag{3}$$

thus its accumulated value is equal to

$$\beta_k = \textstyle\sum_{j=0}^{k} \beta^j. \tag{4}$$

Based on β_k the discounted value of the variables $\{\alpha_{iu}\}$ can be computed for a fixed pair of i and u. Let t_k be the index of the observation in update step k, then we can write up the following recursive formulae for all $n \in \mathcal{I}_u$

$$\alpha_{nu}(t+1) = \frac{s\gamma_{nu}(t) + \beta\alpha_{nu}(t)}{\beta_k}|_{t=t_k} \tag{5}$$

where we used the notation

$$\gamma_{nu}(t) \stackrel{\text{def}}{=} \underbrace{\langle \boldsymbol{\psi}(\hat{r}_{iu}(t)), \boldsymbol{\psi}(\hat{r}_{nu}(t)) \rangle}_{\text{rank kernel}} \underbrace{\langle \boldsymbol{\phi}(\mathbf{q}_i(t)), \boldsymbol{\phi}(\mathbf{q}_n(t)) \rangle}_{\text{item kernel}}. \tag{6}$$

These update formulae are applied whenever the constraint

$$\sum_{n \in \mathcal{I}_u} \alpha_{nu}(t)\gamma_{nu}(t) < 1 - \zeta_i(t) \tag{7}$$

is violated.

IV. EXPERIMENTAL EVALUATION

A. Dataset

We used the FilmTrust (shown by FT) and $100k$ MovieLens (shown by SML) datasets for evaluating our algorithms. We created the FilmTrust dataset by crawling the FilmTrust website on 10th of March 2009 the FilmTrust website[1]. The characteristics of the datasets are shown in table I. The sparsity of a dataset is calculated as follows: $\left(1 - \frac{\textit{non zero entries}}{\textit{all possible entries}}\right)$. The FilmTrust dataset has been used in [5], [6] and MovieLens dataset has been used in [7], [5].

[1] http://trust.mindswap.org/FilmTrust/

B. Metrics

Our specific task in this paper is to predict scores for items that have already been rated by actual users, and to check how well this prediction helps users in selecting high quality items. Considering this, we have used *Mean Absolute Error (MAE)*, *Receiver Operating Characteristic (ROC) sensitivity*, *precision*, *recall*, and *F*1 *measure*. The aim of a recommender system is to minimize MAE and maximize ROC-sensitivity, precision, recall, and F1 metrics. The details of these metrics can be found in [5], [7], [8].

C. Evaluation Methodology

We conducted five-fold cross-validation, where the randomly selected 20% ratings of each user were classified as the test set and the remaining 80% as the training set. We show the average and standard deviation (SD) of the results over the five folds.

V. RESULTS AND DISCUSSION

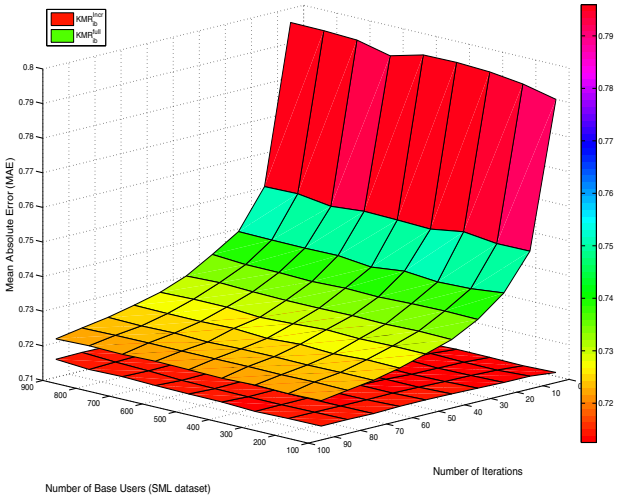

Figure 1. Comparing the performance of the proposed algorithm KMR_{ib}^{incr} with the base one KMR_{ib}^{full} for the MovieLens dataset, when new users are added into the system.

In the following, we denote the algorithm by KMR_{sub}^{super}, where the subscript denotes the variants of the algorithm, which can be item-based (ib) and user-based (ub). The superscript can be $full$ representing the baseline approach where a full iterative model is used to build the model as proposed in [1], $percept$ representing the proposed perceptron algorithm, and $incr$ representing the proposed incremental algorithm.

To check the performance of the KMR^{incr} algorithm, we describe the results for the two related scenarios (1) when new users are introduced into the systems (2) when new movies

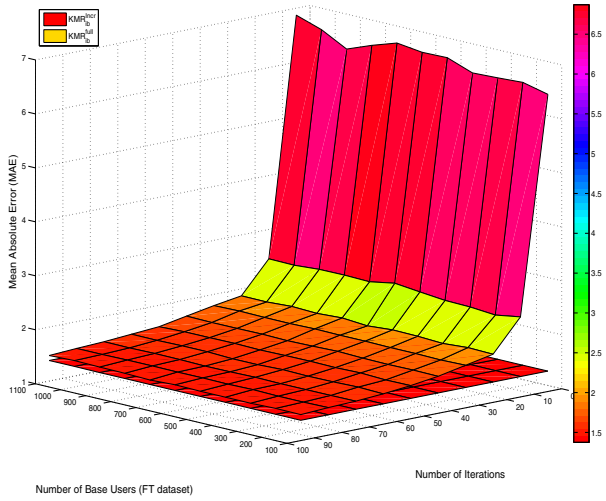

Figure 2. Comparing the performance of the proposed algorithm KMR_{ib}^{incr} with the base one KMR_{ib}^{full} for the FilmTrust dataset, when new users are added into the system.

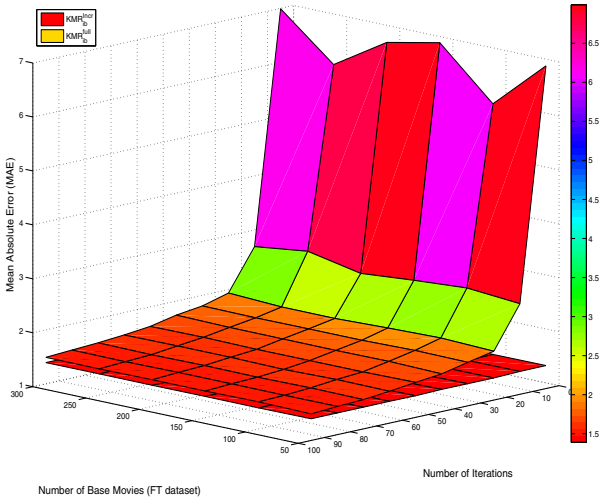

Figure 4. Comparing the performance of the proposed algorithm KMR_{ib}^{incr} with the base one KMR_{ib}^{full} for the FilmTrust dataset, when new movies are added into the system.

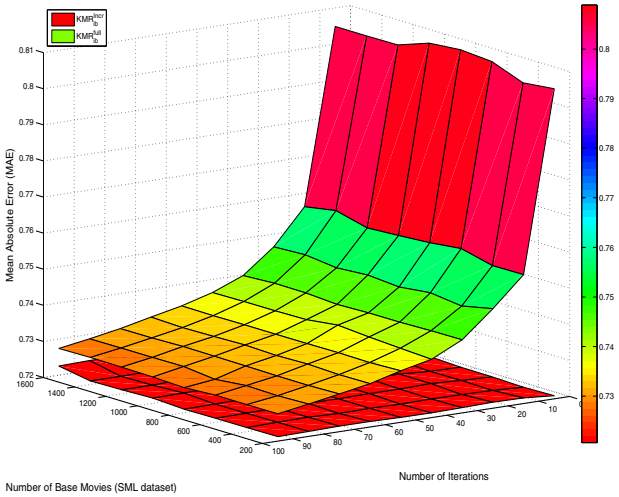

Figure 3. Comparing the performance of the proposed algorithm KMR_{ib}^{incr} with the base one KMR_{ib}^{full} for the MovieLens dataset, when new movies are added into the system.

are introduced into the system. For the MovieLens dataset, we used the time stamp field present in the dataset and sorted the users in the order in which they appear in the system (i.e. in the order in which they made ratings). We then used first $X < M$ users to train the base model, which we call $\mathcal{U}^{base}$ ($\mathcal{U}^{base} \subset \mathcal{U}$), and added the remaining users ($|\mathcal{U}^{new}| = |\mathcal{U}| - |\mathcal{U}^{base}|$) to update the model. Similarly, we sorted the movies in order in which they appear in the system (i.e. movies in the order in which they were rated by the users) and used first $Y < N$ movies to train the base model, which we call $\mathcal{I}^{base}$ ($\mathcal{I}^{base} \subset \mathcal{I}$) and added the remaining movies ($|\mathcal{I}^{new}| = |\mathcal{I}| - |\mathcal{I}^{base}|$) to update the model. We train the base model using the optimal kernel parameters [1] and optimal number of iterations which are found to be 300 for the FilmTrust and 400 for the MovieLens dataset. For the FilmTrust dataset, the test procedures were the same; however we did not sort the users or movies as no time information was available against each rating.

To check the performance of the $KMR^{percept}$, we built the model incrementally for $\mathcal{D}^{Percept} \subset \mathcal{D}$ data points (i.e. we add one data point (rating) at a time), where 20% randomly selected data points of $\mathcal{D}^{Percept}$ were classified as the test set and the remaining as the training set. Again, we sorted the data points in $\mathcal{D}^{Percept}$ based on the time information for the MovieLens dataset.

In this way, we checked the behaviour of the algorithms by simulating the real world behaviour of the recommender systems.

A. Results of the KMR^{incr} algorithm

We compare our algorithm with the baseline approach where we retrain the model from scratch using some fixed number of iterations.

1) New users are added into the system: To check the behaviour of the proposed algorithm when new users enter the system, we performed a series of experiments by changing the base model size from 100 to 943 with a difference of 100 for the SML dataset and from 200 to 1 412 with a difference of 200 for the FT dataset. The $|\mathcal{U}|^{base}$ base users were trained using optimal parameters. The remaining users ($|\mathcal{U}|$-$|\mathcal{U}^{base}|$) were added afterwards into the systems and the model was updated. For each experiment, we changed the number of iterations from 5 to 100 with a difference of 10, keeping the base size parameter fixed, and observed the corresponding MAE.

Table II
COMPARING THE PERFORMANCE OF THE PROPOSED ALGORITHM KMR^{incr} WITH THE BASELINE KMR^{full}, AT BASE USERS SIZE OF 500 ($|\mathcal{U}^{base}|$=500), WHEN NEW USERS ARE ADDED INTO THE SYSTEM. F1 HAS BEEN MEASURED OVER THE TOP 20 RECOMMENDATIONS. THE PERFORMANCE OF THE PROPOSED ALGORITHM IS BETTER THAN (OR COMPARABLE TO) THE BASE ONE.

Algorithm	Itr		MAE		ROC Sensitivity		F1	
	(FT)	(SML)	(FT)	(SML)	(FT)	(SML)	(FT)	(SML)
KMR^{incr}_{ib}	5	5	1.379 ± 0.0013	0.712 ± 0.0003	0.627 ± 0.0014	0.715 ± 0.0014	0.558 ± 0.0007	0.533 ± 0.0004
KMR^{full}_{ib}	300	400	1.380 ± 0.0005	0.713 ± 0.0001	0.629 ± 0.0015	0.715 ± 0.0013	0.564 ± 0.0027	0.531 ± 0.0012
KMR^{incr}_{ub}	5	5	1.387 ± 0.0003	0.735 ± 0.0001	0.666 ± 0.0031	0.725 ± 0.0015	0.595 ± 0.0026	0.529 ± 0.0009
KMR^{full}_{ub}	300	400	1.384 ± 0.0005	0.737 ± 0.0001	0.652 ± 0.0031	0.718 ± 0.0015	0.587 ± 0.0017	0.524 ± 0.0004

Table III
COMPARING THE PERFORMANCE OF THE PROPOSED ALGORITHM KMR^{incr} WITH THE BASELINE KMR^{full}, AT BASE MOVIES SIZE OF 1 000 FOR THE MOVIELENS DATASET ($|\mathcal{I}^{base}|$=1 000) AND 200 FOR THE FILMTRUST DATASET ($|\mathcal{I}^{base}|$=200), WHEN NEW MOVIES ARE ADDED INTO THE SYSTEM. F1 HAS BEEN MEASURED OVER THE TOP 20 RECOMMENDATIONS. THE PERFORMANCE OF THE PROPOSED ALGORITHM IS COMPARABLE TO THE BASE ONE.

Algorithm	Itr		MAE		ROC Sensitivity		F1	
	(FT)	(SML)	(FT)	(SML)	(FT)	(SML)	(FT)	(SML)
KMR^{incr}_{ib}	5	5	1.417 ± 0.0158	0.721 ± 0.0003	0.562 ± 0.0055	0.683 ± 0.0008	0.514 ± 0.0043	0.498 ± 0.0020
KMR^{full}_{ib}	300	400	1.381 ± 0.0021	0.720 ± 0.0001	0.628 ± 0.0037	0.687 ± 0.0011	0.564 ± 0.0037	0.504 ± 0.0006
KMR^{incr}_{ub}	5	5	1.397 ± 0.0010	0.745 ± 0.0004	0.592 ± 0.0100	0.702 ± 0.0013	0.550 ± 0.0060	0.506 ± 0.0017
KMR^{full}_{ub}	300	400	1.382 ± 0.0003	0.742 ± 0.0001	0.651 ± 0.0011	0.705 ± 0.0008	0.588 ± 0.0004	0.507 ± 0.0009

Figures 1 and 2 show the performance of the proposed algorithm with the baseline for the MovieLens and the FilmTrust datasets respectively. Figures 1 and 2 show that the proposed algorithm outperforms the baseline approach at every combination of base model size and number of iterations.

Table II compares the performance of the proposed algorithm at model size[2] of 500 with the baseline approach. It must be noted that for the baseline approach shown in table II, the solution is found using the optimal number of iterations, which is very expensive compared to the proposed algorithm. Table II shows that the proposed approach gives comparable results to the baseline one.

2) New movies are added into the system: To check the behaviour of the proposed algorithm when new movies enter into the system, we performed a series of experiments by changing the base model size from 200 to 1 682 with a difference of 200 and from 50 to 314 with a difference of 50 for the FT dataset. The $|\mathcal{I}^{base}|$ base movies were trained using the optimal parameters and 400 number of iterations. The remaining movies ($|\mathcal{I}| - |\mathcal{I}^{base}|$) were added afterwards into the systems and the model was updated. For each experiment, we changed the number of iterations from 5 to 95 with the difference of 10, keeping the base size fixed, and observed the corresponding MAE.

Figures 3 and 4 show the performance of the proposed algorithm with the baseline for the MovieLens and FilmTrust datasets respectively. We observe that the proposed algorithm outperforms the baseline approach at each combination of base model size and number of iterations.

Table III compares the performance of the proposed algorithm at model size of 1000 for the MovieLens and 200 for the FilmTrust dataset with the baseline approach tuned using the optimal number of iterations. Again, we observe that the performance of the proposed algorithm is comparable to the base one.

[2]This model size was chosen as an example. Similar results were observed for the other sizes as well.

B. Results of the $KMR^{percept}$ algorithm

We built the models for the following values of $\mathcal{D}^{percept}$: (1) $\{1\,000,\ 2\,000,\ 5\,000,\ 10\,000,\ 25\,000,\ 50\,000,\ 100\,000\}$ for the SML dataset and (2) $\{1\,000,\ 2\,000,\ 5\,000,\ 10\,000,\ 15\,000,\ 20\,000,\ 25\,730\}$ for the FT dataset. The results of the proposed algorithm at varying values of $\mathcal{D}^{percept}$ have been compared with the baseline approach—the one proposed in [1] employing the iterative model to build the model using optimal parameters—and have been shown in figure 5. Figure 5 shows that the proposed algorithm gives comparable MAE as compared to the baseline approach.

Table IV compares the performance of $KMR^{percept}$ with KMR^{full} at sample size of $10,000$ (i.e. $\mathcal{D}^{percept} = 10\,000$). We observe that the proposed algorithm gives comparable results to the baseline one in terms of accuracy and F1 metrics.

VI. CONCLUSION

Kernel Mapping Recommender (KMR) algorithms are a new class of kernel-based methods for solving the recommender system problem that offer the state-of-the-art performance. Although KMR algorithms have the potential to build the model using the offline stage; however, they have to recompute the model on the arrival of new data that is costly both in terms of computation time and storage, making this class of algorithm unsuitable for modern e-commerce systems where data are being added continuously in the system.

In this paper, we have proposed two algorithms, namely $KMR^{percept}$ and KMR^{incr} that solve the aforementioned problem. The results show that the KMR^{incr} algorithm updates the model on the arrival of the new data and maintains a good level of accuracy. Similarly, the $KMR^{percept}$ algorithm is accurate and scales well with the data as it has the potential to build the model by sequentially processing the dataset.

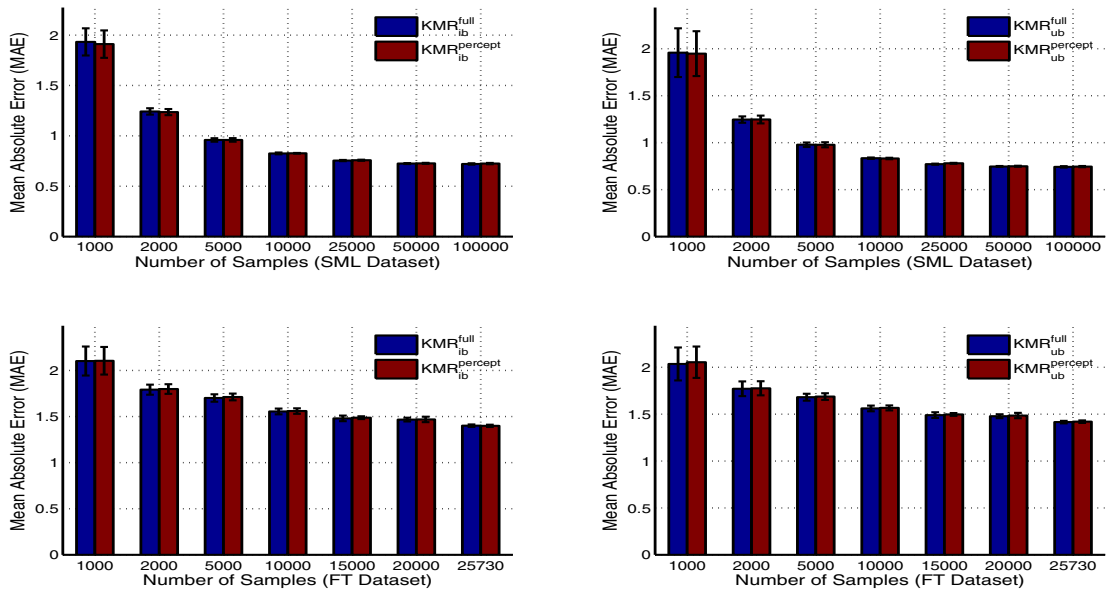

Figure 5. Comparing the performance of the proposed algorithm $KMR^{percept}$ with the baseline KMR^{full} under various values of $\mathcal{D}^{Percept}$. The baseline approach is trained using 400 and 300 iterations for the MovieLens and FilmTrust datasets respectively.

Table IV
COMPARING THE PERFORMANCE OF THE PROPOSED ALGORITHMS $KMR^{percept}$ WITH THE BASE ONE KMR^{full} AT A SAMPLE SIZE OF 10 000 ($\mathcal{D}^{Percept} = 10\,000$). F1 HAS BEEN MEASURED OVER THE TOP 20 RECOMMENDATIONS. THE PERFORMANCE OF THE PROPOSED ALGORITHM IS BETTER THAN (OR COMPARABLE TO) THE BASE ONE.

Algorithm	**MAE**		**ROC Sensitivity**		**F1**	
	(FT)	**(SML)**	**(FT)**	**(SML)**	**(FT)**	**(SML)**
KMR_{ib}^{full}	1.468 ± 0.0298	0.826 ± 0.0038	0.549 ± 0.0197	0.607 ± 0.0152	0.512 ± 0.0197	0.471 ± 0.0173
$KMR_{ib}^{percept}$	1.466 ± 0.0313	0.825 ± 0.0089	0.547 ± 0.0193	0.599 ± 0.0117	0.510 ± 0.0199	0.499 ± 0.0194
KMR_{ub}^{full}	1.485 ± 0.0270	0.831 ± 0.0080	0.613 ± 0.0091	0.626 ± 0.0241	0.556 ± 0.0141	0.472 ± 0.0216
$KMR_{ub}^{percept}$	1.478 ± 0.0305	0.834 ± 0.0073	0.596 ± 0.0125	0.615 ± 0.0343	0.548 ± 0.0162	0.467 ± 0.0231

As a future work, we would compare the proposed algorithms with online matrix factorization techniques [9] over large datasets such as Netflix [10].

ACKNOWLEDGMENT

The second author has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 (Specific Programme Cooperation, Theme 3,Information and Communication Technologies) under grant agreement no. 270273, Xperience.

REFERENCES

[1] S. Ghazanfar, M.A. Szedmak and A. Prugel-Bennett, "Kernel mapping recommender systems," *Information Sciences*.

[2] A. T. Gediminas Adomavicius, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, pp. 734–749, 2005.

[3] M. Ghazanfar and A. Prugel-Bennett, "The advantage of careful imputation sources in sparse data-environment of recommender systems: Generating improved svd-based recommendations," in *IADIS European Conference on Data Mining*, July 2011.

[4] R. Burke, "Hybrid recommender systems: Survey and experiments," *User Modeling and User-Adapted Interaction*, vol. 12, no. 4, pp. 331–370, November 2002.

[5] M. Ghazanfar and A. Prugel-Bennett, "Building Switching Hybrid Recommender System Using Machine Learning Classifiers and Collaborative Filtering," *IAENG International Journal of Computer Science*, vol. 37, no. 3, pp. 272–287, 2010.

[6] ——, "Fulfilling the needs of gray-sheep users in recommender systems, a clustering solution," in *2011 International Conference on Information Systems and Computational Intelligence*, January 2011. [Online]. Available: http://eprints.ecs.soton.ac.uk/21770/

[7] ——, "A scalable, accurate hybrid recommender system," in *The 3rd International Conference on Knowledge Discovery and Data Mining (WKDD 2010)*. IEEE, 9–10 January, 2010, Thailand, 2010. [Online]. Available: http://eprints.ecs.soton.ac.uk/18430/

[8] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Application of dimensionality reduction in recommender system–a case study," in *ACM WebKDD 2000 Web Mining for E-Commerce Workshop*. Citeseer, 2000.

[9] S. Rendle and L. Schmidt-Thieme, "Online-updating regularized kernel matrix factorization models for large-scale recommender systems," in *Proceedings of the 2008 ACM conference on Recommender systems*. ACM, 2008, pp. 251–258.

[10] J. Bennett and S. Lanning, "The netflix prize," in *Proceedings of KDD Cup and Workshop*, vol. 2007. Citeseer, 2007.